

Thursday, February 27<sup>th</sup>, 2025

Simon Scheidegger (UNIL)

# Outline

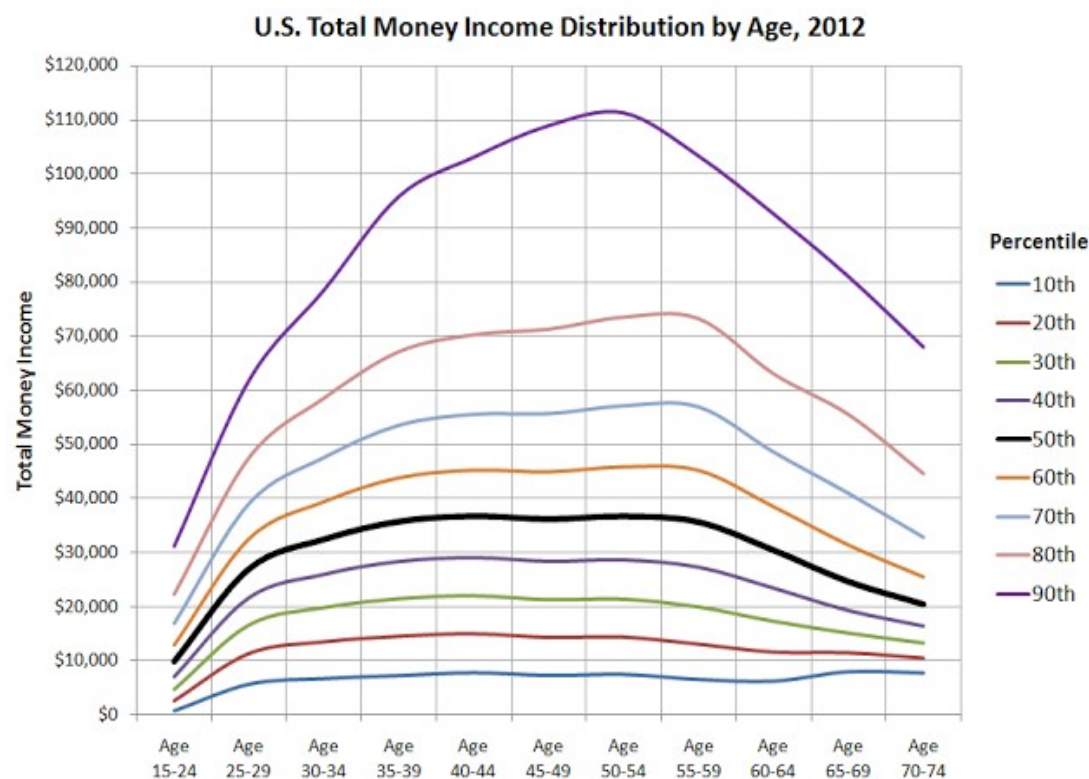
- 1. Motivation – why are we doing what we are doing**
- 2. Technical background of “Deep Equilibrium Nets”**
  2. I. From artificial neural networks to “Deep Equilibrium Nets”.
  2. II. A simple benchmark model (Brock & Mirman (1972))  
Notebook: `code/01_Brock_Mirman_1972_DEQN.ipynb`
  2. III. \*A simple benchmark OLG model (take-home slides due to limited time).

# Humans are all different – heterogeneous



- **Heterogeneity** a crucial ingredient in contemporary models:
  - to study e.g. cross-sectional consumption response to aggregate shocks.
  - to model, e.g., social security.
- Example OLG models:
  - How many age groups?
  - borrowing constraints?
  - aggregate shocks?
  - idiosyncratic shocks?
  - liquid / illiquid assets\*\*?

→ **Models: heterogeneous & high-dimensional**



\*\*see, e.g., Kaplan et al. (2018), Wong (2018),...

# Dynamic Stochastic Models

e.g. Judd (1998), Ljungquist & Sargent (2004),...

$$\mathbb{E} [E(\mathbf{x}_t, \mathbf{x}_{t+1}, p(\mathbf{x}_t), p(\mathbf{x}_{t+1})) | \mathbf{x}_t, p(\mathbf{x}_t)] = 0$$

$$\mathbf{x}_{t+1} \sim \mathcal{P}(\cdot | \mathbf{x}_t, p(\mathbf{x}_t))$$

**x**: point in state space; describes your system.

State-space potentially **irregularly-shaped** and **high-dimensional**.

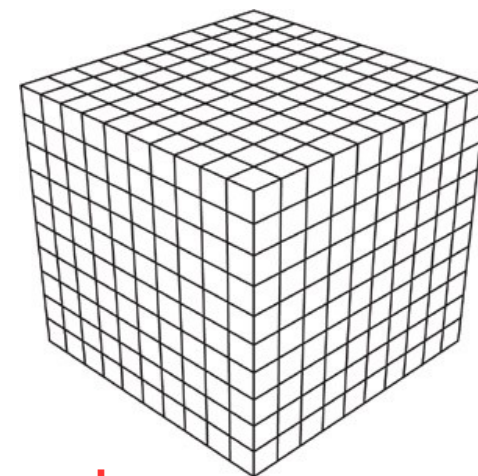
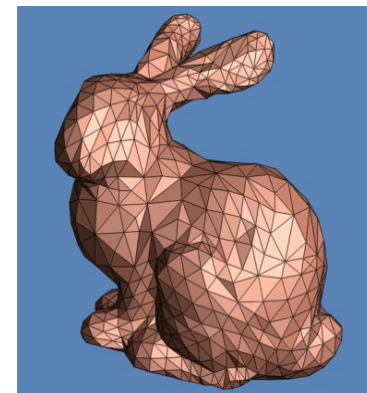
**p**: time-invariant policy function.

“old solution”:  
high-dimensional functions on which **we interpolate**.

→ **N<sup>d</sup>** points in ordinary discretization schemes.

→ **“Curse of dimensionality”**.

→ Usually: solve many **non-linear systems of equations** by **invoking a solver**.



# What is high-dimensional?

<u>#State Variables (Dimensions)</u>	<u>#Points</u>	<u>Time-to-solution</u>
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
...	...	...
20	1e20	3 trillion years (240x age of the universe)

## **Dimension reduction**

*Exploit symmetries, e.g., via the active subspace method*

## **Deal with #Points**

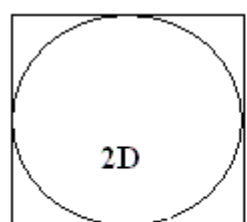
*e.g., via (Smolyak/adaptive) sparse grids*

## **High-performance computing**

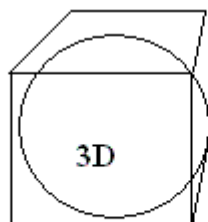
*Reduces time to solution, but not the problem size*

# Volumes in high dimensions

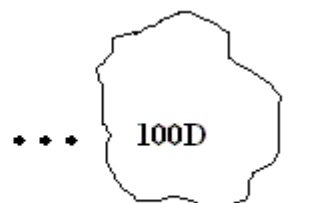
- Consider a cube of unit lengths containing a sphere of unit radius in higher dimensions.
- For large dimensions: ratio **Volume(Sphere)/Volume(Cube)  $\rightarrow 0$**



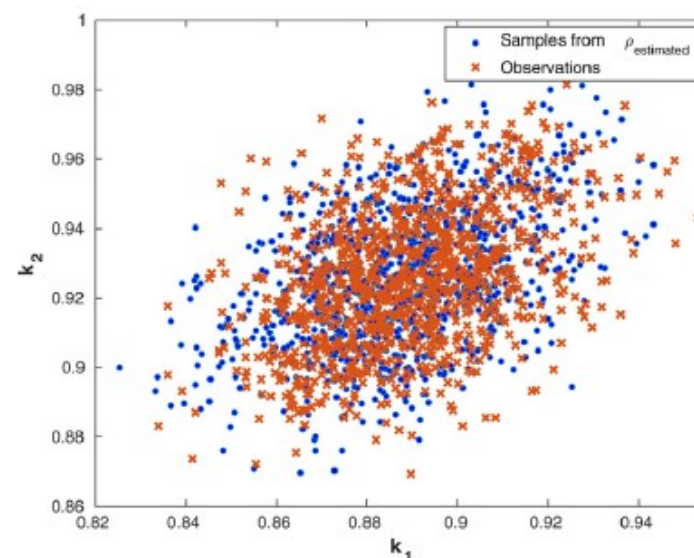
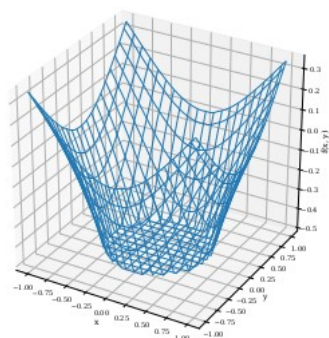
ratio:  $4/\pi = 1.27$



ratio:  $6/\pi = 1.91$



ratio:  $4.2 \cdot 10^{39}$



Scheidegger & Billionis (2019)

# Abstract Problem Formulation

- i) Contemporary dynamic models: heterogeneous & high-dimensional
- ii) Want to compute global solutions to dynamic stochastic models with high-dimensional state spaces
  - Have to **approximate** and **interpolate** high-dimensional functions on irregular-shaped geometries
  - Problem: curse of dimensionality
- iii) **Want to alleviate the curse of dimensionality**
- iv) **Want locality of approximation scheme**
- v) **Speed-up** → potentially access contemporary HPC systems

# Our solution: Deep Equilibrium Nets

- Solving, e.g., rich OLG models numerically is a **formidable task**.
- Models are often formulated in a **stylized** fashion to remain computationally **tractable**.
- We develop a **generic solution framework** based on **neural networks** to solve highly-complex dynamic stochastic models.

## Key ideas:

1. Use the **the implied error in the optimality conditions**, as **loss function**.
2. Learn the equilibrium functions with **stochastic gradient descent**.
3. Take the (training) **data points from a simulated path**
  - can be generated at **virtual zero cost**.



## 2. Technical Background



# Recall: what is a deep neural network

see, e.g., Cybenko (1989), Hornik (1991)

- Neural networks are **flexible function approximators**.
- A neural net is characterized by its **parameters  $\boldsymbol{\rho}$** .
- Given a **parameter vector  $\boldsymbol{\rho}$**  and **input vector  $\mathbf{x}$** , denote the neural net as  $\mathcal{N}_{\boldsymbol{\rho}}$ , and some desired function with  $\mathbf{f}$ .

$$\begin{aligned}\mathcal{N}_{\boldsymbol{\rho}} &: \mathbb{R}^{N_{\text{in}}} \rightarrow \mathbb{R}^{N_{\text{out}}} : \mathbf{x} \rightarrow \mathcal{N}_{\boldsymbol{\rho}}(\mathbf{x}) \\ \mathbf{f}(\mathbf{x}) &: \mathbb{R}^{N_{\text{in}}} \rightarrow \mathbb{R}^{N_{\text{out}}} : \mathbf{x} \rightarrow \mathbf{f}(\mathbf{x})\end{aligned}$$

- We desire parameters  $\boldsymbol{\rho}$ , such that

$$\|\mathcal{N}_{\boldsymbol{\rho}} - \mathbf{f}\|_{\text{some norm}} = 0$$

# What is a deep neural network?

Consider:

$$\begin{aligned} \text{input} &:= \mathbf{x} \rightarrow W_{\rho}^1 \mathbf{x} + \mathbf{b}_{\rho}^1 =: \text{hidden 1} \\ &\rightarrow \text{hidden 1} \rightarrow W_{\rho}^2(\text{hidden 1}) + \mathbf{b}_{\rho}^2 =: \text{hidden 2} \\ &\rightarrow \text{hidden 2} \rightarrow W_{\rho}^3(\text{hidden 2}) + \mathbf{b}_{\rho}^3 =: \text{output} \end{aligned}$$

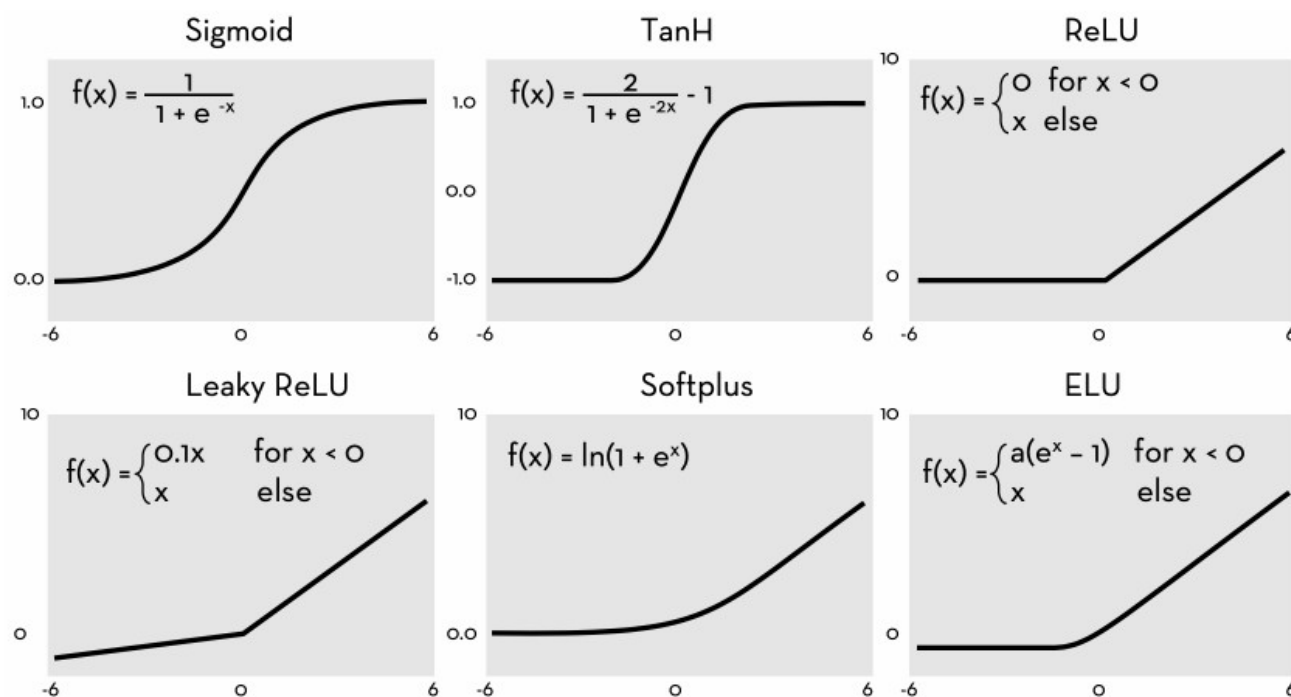
The **parameters**  $\rho$  are the entries of the matrices  $(W_{\rho}^1, W_{\rho}^2, W_{\rho}^3)$  and vectors  $(\mathbf{b}_{\rho}^1, \mathbf{b}_{\rho}^2, \mathbf{b}_{\rho}^3)$

# What is a deep neural network?

So far we have a **concatenation of affine maps** and therefore an affine map.

Next ingredient: **activation functions**  $\phi^1, \phi^2, \phi^3$

Popular activation functions are:



# What is a deep neural network?

Now we obtain:

$$\begin{aligned} \text{input} &:= \mathbf{x} \rightarrow \phi^1(W_{\rho}^1 \mathbf{x} + \mathbf{b}_{\rho}^1) =: \text{hidden 1} \\ &\rightarrow \text{hidden 1} \rightarrow \phi^2(W_{\rho}^2(\text{hidden 1}) + \mathbf{b}_{\rho}^2) =: \text{hidden 2} \\ &\rightarrow \text{hidden 2} \rightarrow \phi^3(W_{\rho}^3(\text{hidden 2}) + \mathbf{b}_{\rho}^3) =: \text{output} \end{aligned}$$

The **neural net** is then given by the choice of **activation functions** and the **parameters  $\rho$** .

# How to find good parameters $\rho$ ?

## The standard way:

Step 1: get “labelled data”  $\mathcal{D} := \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_{|\mathcal{D}|}, \mathbf{y}_{|\mathcal{D}|})\}$  where  $\mathbf{y}_i = \mathbf{f}(\mathbf{x}_i)$  is the correct output  $\rightarrow$  **Supervised learning**

Step 2: Define a **loss function**, for example:

$$l_{\rho} := \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}} (\mathbf{y}_i - \mathcal{N}_{\rho}(\mathbf{x}_i))^2$$

Step 3: Adjust the parameters to minimize the loss via (**stochastic gradient descent**):

$$\rho_i^{\text{new}} = \rho_i^{\text{old}} - \alpha^{\text{step}} \frac{\partial l_{\rho^{\text{old}}}}{\partial \rho_i^{\text{old}}}$$

the step-width  $\alpha^{\text{step}}$  is called the “**learning rate**” and the process of adjusting the parameters is called “learning”.

# How to find good parameters $\rho$ ?

- The **deeper and larger** the neural net becomes, the **more flexible** it is as a function approximator, ...  
  
... but the **more data** it will need  
→ rule of thumb: **#observations/parameters** ~ 10x (Marsland (2014)).
- In economic applications, the **standard way of solving for equilibria** is to use time iteration → **“small” data**.  
(e.g., Sparse Grids (Brumm & Scheidegger (2017), Krüger & Kübler (2004), Judd et al. (2014))

# The issue with time iteration

- **Time Iteration – collocation** (see, e.g., Judd (1998), and references therein)

1. Select a grid  $G$ , and a policy function  $f^{\text{start}}$ . Set  $f^{\text{next}} \equiv f^{\text{start}}$ .
2. Make one time iteration step:
  1. For all  $g \in G$ , find  $f(g)$  that solves the Period-to-Period Equilibrium Problem given  $f^{\text{next}}$ .
  2. Use solutions at all grid points  $G$  to interpolate  $f$  (how?).
3. Check error criterion: If  $\|f - f^{\text{next}}\|_{\infty} < \epsilon$ , report solution:  $\tilde{f} = f$ . Else set  $f^{\text{next}} \equiv f$  and go to step 2.

- Solving non-linear sets of equations in every iteration can be a **daunting task**.
- What if non-linear **solver does not converge**? Construction of approximator may crash.
- Can the dynamic economic **problem** at hand **be mapped onto a grid**?



# An “economic” loss function

- **Novelty**: we propose an “economic” loss function:

$$l_{\rho} := \frac{1}{N_{\text{path length}}} \sum_{\mathbf{x}_i \text{ on sim. path}} (\mathbf{G}(\mathbf{x}_i, \mathcal{N}_{\rho}(\mathbf{x}_i)))^2$$

where we use  $\mathcal{N}_{\rho}$  to simulate a path.

- $\mathbf{G}$  is chosen such that the **true equilibrium policy  $\mathbf{f}(\mathbf{x})$**  is defined by

$$\mathbf{G}(\mathbf{x}, \mathbf{f}(\mathbf{x})) = 0 \quad \forall \mathbf{x}.$$

- **$\mathbf{G}(\cdot, \cdot)$ : implied error in the optimality conditions** (unit-free Euler errors)
- Therefore, there is **no need for labels** to evaluate our loss function.  
→ **Unsupervised Machine Learning**.

# Training Deep Equilibrium Nets

**Algorithm 1:** Algorithm for training deep equilibrium nets.

**Data:**

$T$  (length of an episode),  
 $N^{\text{epochs}}$  (number of epochs on each episode),  
 $\tau^{\text{max}}$  (desired threshold for max error),  
 $\tau^{\text{mean}}$  (desired threshold for mean error),  
 $\epsilon^{\text{mean}} = \infty$  (starting value for current mean error),  
 $\epsilon^{\text{max}} = \infty$  (starting value for current max error),  
 $N^{\text{iter}}$  (maximum number of iterations),  
 $\rho^0$  (initial parameters of the neural network),  
 $\mathbf{x}_1^0$  (initial state to start simulations from),  
 $i = 0$  (set iteration counter),  
 $\alpha^{\text{learn}}$  (learning rate)

**Result:**

success (boolean if thresholds were reached)

$\rho^{\text{final}}$  (final neural network parameters)

**while**  $((i < N^{\text{iter}}) \wedge ((\epsilon^{\text{mean}} \geq \tau^{\text{mean}}) \vee (\epsilon^{\text{max}} \geq \tau^{\text{max}})))$  **do**

$\mathcal{D}_{\text{train}}^i \leftarrow \{\mathbf{x}_1^i, \mathbf{x}_2^i, \dots, \mathbf{x}_T^i\}$  (generate new training data by simulating an episode of  $T$  periods as implied by the parameters  $\rho^i$ )

$\mathbf{x}_0^{i+1} \leftarrow \mathbf{x}_T^i$  (set new starting point)

$\epsilon^{\text{max}} \leftarrow \max \left\{ \max_{\mathbf{x} \in \mathcal{D}_{\text{train}}^i} |e_{\mathbf{x}}^{\cdot\cdot\cdot}(\rho)| \right\}$  (calculate max error on new data)

$\epsilon^{\text{mean}} \leftarrow \max \left\{ \frac{1}{T} \sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}^i} |e_{\mathbf{x}}^{\cdot\cdot\cdot}(\rho)| \right\}$  (calculate mean error on new data)

**for**  $j \in [1, \dots, N^{\text{epochs}}]$  **do**

        (learn  $N^{\text{epochs}}$  on data)

**for**  $k \in [1, \dots, \text{length}(\rho)]$  **do**

$$\rho_k^{i+1} = \rho_k^i - \alpha^{\text{learn}} \frac{\partial \ell_{\mathcal{D}_{\text{train}}^i}(\rho^i)}{\partial \rho_k^i}$$

            (do a gradient descent step to update the network parameters)

**end**

**end**

$i \leftarrow i + 1$  (update episode counter)

**end**

**if**  $i = N^{\text{iter}}$  **then return** (success  $\leftarrow \text{False}$ ,  $\rho^{\text{final}} \leftarrow \rho^i$ );

**else return** (success  $\leftarrow \text{True}$ ,  $\rho^{\text{final}} \leftarrow \rho^i$ );

## 2. II. A simple benchmark model

- Lets check-out the notebook:  
code/01\_Brock\_Mirman\_1972\_DEQN.ipynb

### Simple Introduction to Deep Equilibrium Nets

#### Notebook 1: no uncertainty and exogenous sampling of states

Notebook by [Marlon Azinovic](#), [Luca Gaegauf](#), and [Simon Scheidegger](#), August 2023.

#### Purpose of the notebook and economic model

The notebook should serve as a simple introduction to [Deep Equilibrium Nets](#), a deep learning based method introduced in [Azinovic et al. \(2022\)](#). To focus on the method, we are going to solve a simple optimal growth model with one representative agent, a simplified version of [Brock and Mirman \(1972\)](#).

The planner aims to maximize her time-separable life time utility subject to her budget constraint:

$$\begin{aligned} \max_{\{c_t\}_{t=0}^{\infty}} \quad & \sum_{t=0}^{\infty} \beta^t \ln(C_t) \\ \text{s.t.} \quad & K_{t+1} + C_t = Y_t + (1 - \delta)K_t \end{aligned}$$

where  $Y_t = K_t^\alpha$ .

When we assume full depreciation, i.e.  $\delta = 1$ , this particular problem has an analytical solution:

$$K_{t+1} = \beta \alpha K_t^\alpha$$

We can numerically solve the above planner's problem by using any global solution algorithm such as the value function iteration or the time iteration collocation.

However in this notebook, we demonstrate how the recursive equilibrium can be directly approximated by the deep neural net following [Azinovic et al. \(2022\)](#).

“To pull a bigger wagon, it is easier to add more oxen than to grow a gigantic ox”

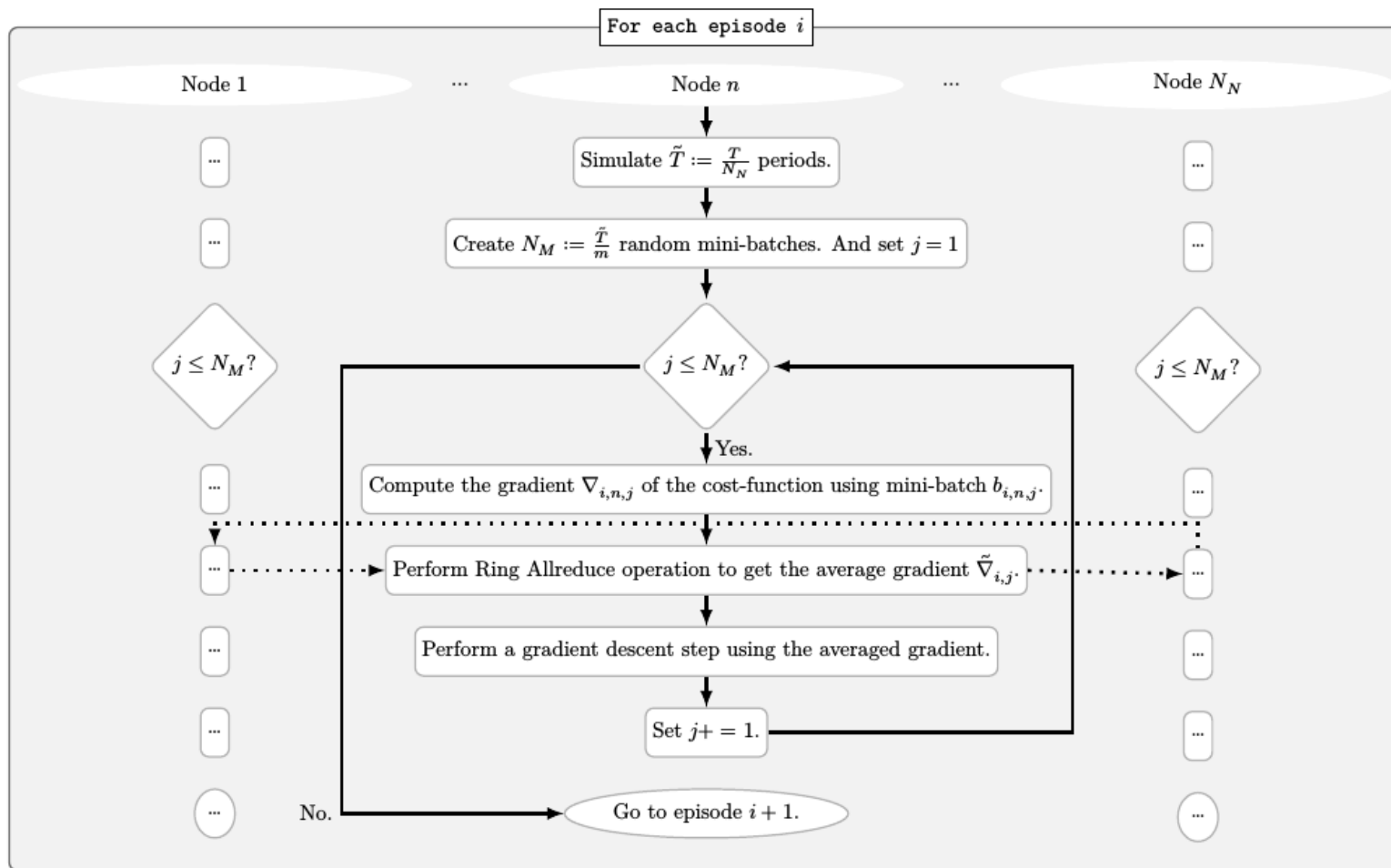
(Skjellum et al. 1999)





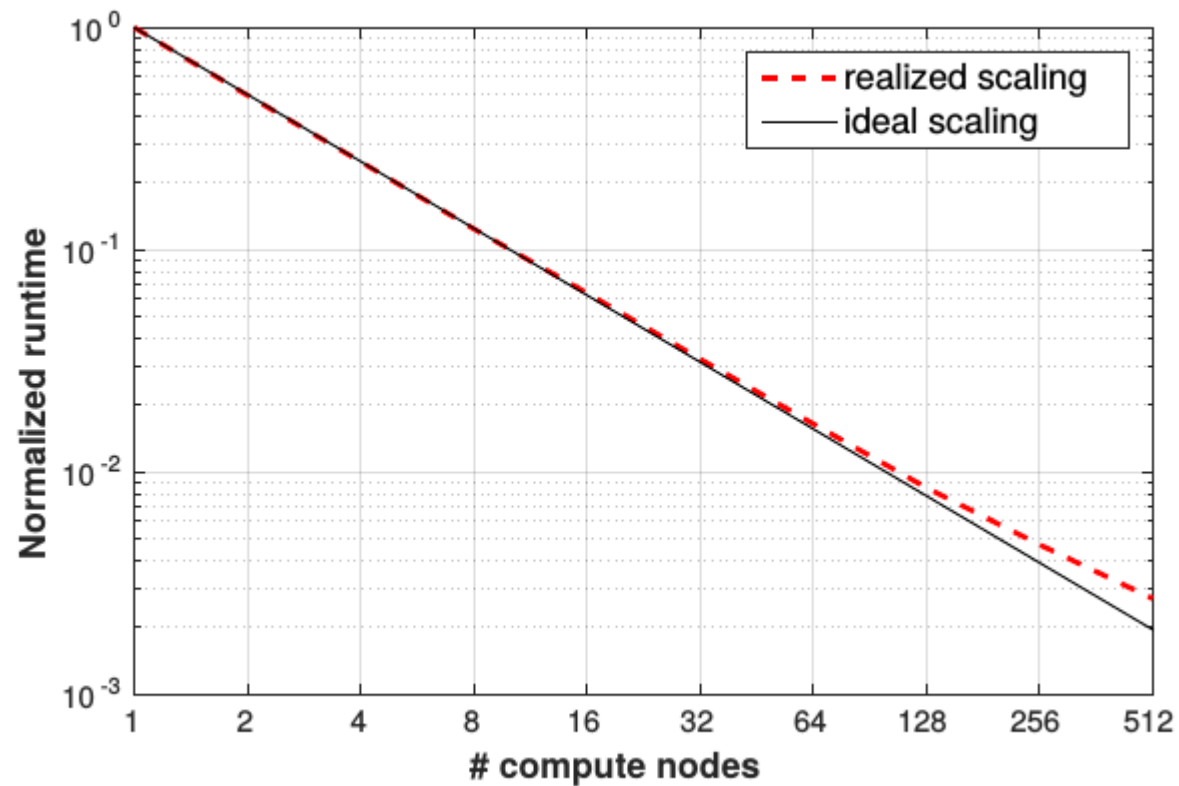
- We use Horovod to parallelize DEQ.
- Based on MPI.
- Idea: **use data-parallelism.**
  - The **generation of training data** is **divided across compute nodes.**
  - **Each node computes the gradient of the cost-function** concerning the parameters of the neural network on it's given batch of data.
  - Then, all nodes are synchronized and the **gradient descent step** is taken using the **average gradient.**

# Parallelization Scheme



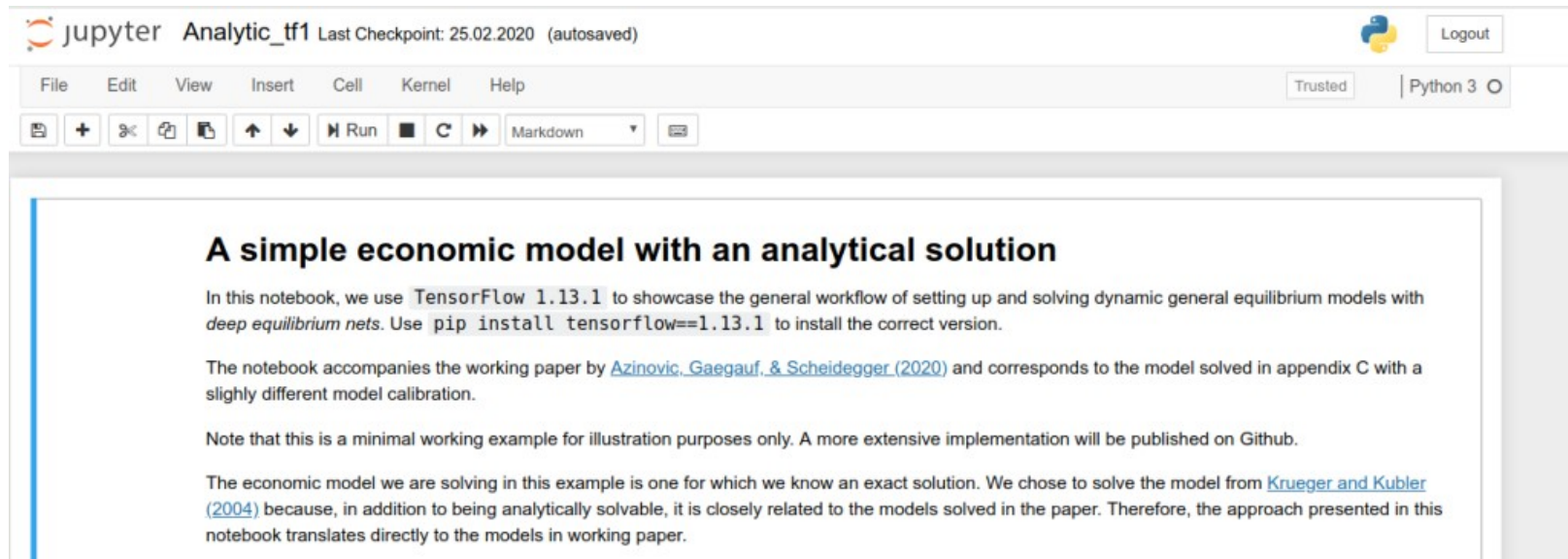
# Scalability on “Piz Daint” (CSCS)

- Excellent strong scaling efficiency (over 70% at 512 nodes).



# More DEQN codes available

Download it here: <https://github.com/sischei/DeepEquilibriumNets>



**A simple economic model with an analytical solution**

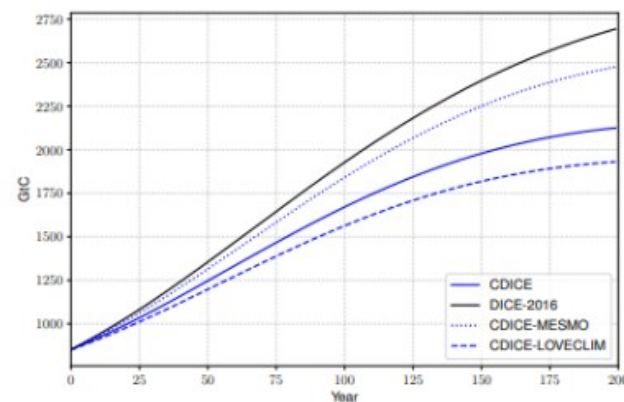
In this notebook, we use TensorFlow 1.13.1 to showcase the general workflow of setting up and solving dynamic general equilibrium models with *deep equilibrium nets*. Use `pip install tensorflow==1.13.1` to install the correct version.

The notebook accompanies the working paper by [Azinovic, Gaegauf, & Scheidegger \(2020\)](#) and corresponds to the model solved in appendix C with a slightly different model calibration.

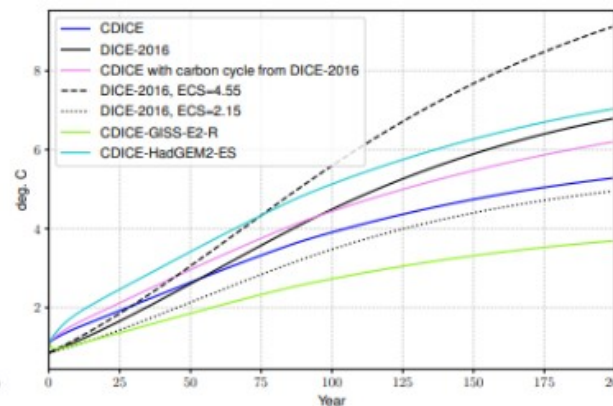
Note that this is a minimal working example for illustration purposes only. A more extensive implementation will be published on Github.

The economic model we are solving in this example is one for which we know an exact solution. We chose to solve the model from [Krueger and Kubler \(2004\)](#) because, in addition to being analytically solvable, it is closely related to the models solved in the paper. Therefore, the approach presented in this notebook translates directly to the models in working paper.

## The Climate in Climate Economics



(a) Mass of carbon in the atmosphere, BAU case

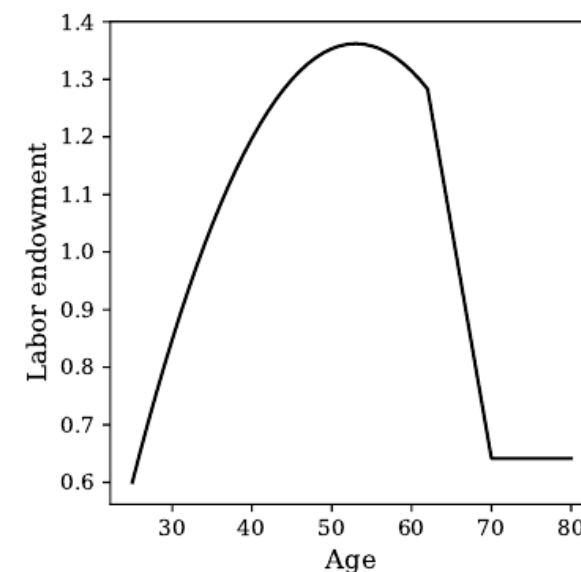


(b) Temperature of the atmosphere, BAU case



## 2.III. A benchmark OLG model (potentially take-home material)

- Time is discrete:  $t = 0, \dots, \infty$
- Agents live for **N periods** (N=60 years).
- **One representative household per cohort.**
- Every  $t$ , **a representative household is born.**
- **No uncertainty** about lifetime.
- There are **exogenous aggregate shocks  $z$**  that follow a Markov chain.
- Each period, the agents alive receive a strictly positive **labour endowment** which depends on the **age of the agent** alone.



# Households

- Household supplies its labour endowment **inelastically** for a market wage  **$w_t$** .
- Agents (with current age  **$s$** ) alive **maximize their remaining** time-separable discounted expected **lifetime utility** ( $\beta < 1$ ):

$$\sum_{i=0}^{N-s} E_t [\beta^i u(c_{t+i}^{s+i})] \quad (1)$$

- Households can **save a unit of consumption good** to obtain a unit of **capital good** next period (denoted as  $a_t^s$ ).
- The **savings** will become **capital in the next period**:

$$a_t^s = k_{t+1}^{s+1}, \forall t, \forall s \in \{1, \dots, N-1\} \quad (2)$$

# Households (II)

- Households cannot die with debt.
- Borrowing is allowed up to an exogenously given level:  $a_t^s \geq \underline{a}$ . (3)
- At time  $t$ , the households sell their capital to the firm at market price  $r_t > 0$ .
- The budget constraint of the household  $s$  in period  $t$  is

$$c_t^s + a_t^s = r_t k_t^s + l_t^s w_t \quad (4)$$

- The agents are born, and die without any assets  $k_t^1 = 0$  and  $a_t^N = 0$

# Firms & Markets

- There is a single **representative firm** with **Cobb-Douglas** production.
- The **total factor productivity**  $\eta$  (**TFP**) and the **depreciation**  $\delta$  depend on the **exogenous shock**  $z$  alone ( $\eta(z) \in \{0.85, 1.15\}$ ,  $\delta(z) \in \{0.5, 0.9\}$ )

$$\pi^\delta = \begin{bmatrix} 0.98 & 0.02 \\ 0.25 & 0.75 \end{bmatrix}, \quad \pi^\eta = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \quad z^\delta \otimes z^\eta = z \in \{0, 1, 2, 3\}$$

- Each period, **after the shock has realized**, the **firm buys capital** and **hires labour** to **maximize its profits**, taking prices as given.
- The **stochastic production** function is given by

$$f(K, L, z) = \eta(z)K^\alpha L^{1-\alpha} + K(1 - \delta(z))$$

- There are **competitive spot markets** for **consumption, capital, labor**.

# Equilibrium

**Definition 1 (competitive equilibrium)** A competitive equilibrium, given initial conditions  $z_0, \{k_0^s\}_{s=1}^{N-1}$ , is a collection of choices for households  $\{(c_t^s, a_t^s)_{s=1}^N\}_{t=0}^\infty$  and for the representative firm  $(K_t, L_t)_{t=0}^\infty$  as well as prices  $(r_t, w_t)_{t=0}^\infty$ , such that

1. Given  $(r_t, w_t)_{t=0}^\infty$ , the choices  $\{(c_t^s, a_t^s)_{s=1}^N\}_{t=0}^\infty$  maximize (1), subject to (2), (3), and (4).
2. Given  $r_t, w_t$ , the firm maximizes profits, i.e.,

$$(K_t, L_t) \in \arg \max_{K_t, L_t \geq 0} f(K_t, L_t, z_t) - r_t K_t - w_t L_t.$$

3. All markets clear: For all  $t$

$$L_t = \sum_{s=1}^N l_t^s,$$

$$K_t = \sum_{s=1}^N k_t^s,$$

- (1): max. remaining lifetime utility  
 (2): savings  $\rightarrow$  capital in next period  
 (3): borrowing constraint.  
 (4): budget constraint.

# Equilibrium Conditions

- The first order conditions of the **firms maximization problem** imply

$$w(z^t) = (1 - \alpha)\eta(z_t) K(z^t)^\alpha L(z^t)^{-\alpha}$$

$$r(z^t) = \alpha\eta(z_t) K(z^t)^{\alpha-1} L(z^t)^{1-\alpha} + (1 - \delta(z_t))$$

- Optimality conditions **for any given generation of age  $s \in 1, \dots, N - 1$ :**

$$u'(c^s(z^t)) = \beta E_{z_t} [u'(c^{s+1}(z^t, z_{t+1})) r(z^t, z_{t+1})] + \lambda^s(z^t)$$

$$\lambda^s(z^t) \cdot (a^s(z^t) - \underline{a}) = 0$$

$$a^s(z^t) - \underline{a} \geq 0$$

$$\lambda^s(z^t) \geq 0$$

- The generation of **terminal age**  $N$  simply **consumes everything** it has.

# Recall OLG: an explicit cost function

$$\ell_{\mathcal{D}_{\text{train}}}(\rho) := \frac{1}{|\mathcal{D}_{\text{train}}|} \frac{1}{N-1} \sum_{\mathbf{x}_j \in \mathcal{D}_{\text{train}}} \sum_{i=1}^{N-1} \left( (e_{\text{REE}}^i(\mathbf{x}_j))^2 + (e_{\text{KKT}}^i(\mathbf{x}_j))^2 \right)$$

$$e_{\text{REE}}^i(\mathbf{x}_j) := \frac{u'^{-1} \left( \beta \mathbb{E}_{z_j} \left[ r(\hat{\mathbf{x}}_{j,+}) u'(\hat{c}^{i+1}(\hat{\mathbf{x}}_{j,+})) \right] + \hat{\lambda}^i(\mathbf{x}_j) \right)}{\hat{c}^i(\mathbf{x}_j)} - 1$$

$$e_{\text{KKT}}^i(\mathbf{x}_j) := \hat{\lambda}^i(\mathbf{x}_j) (\hat{a}^i(\mathbf{x}_j) - \underline{a})$$

$$\hat{\mathbf{x}}_{j,+} = \begin{bmatrix} z_+ \\ 0 \\ \hat{a}^{[1:N-1]}(\mathbf{x}_j) \end{bmatrix} \longrightarrow \text{Sampling from the relevant states}$$

# Functional Rational Expectations Equilibrium

See, e.g., Spear (1988)

**FREE:** A function mapping states to policies that are consistent with the equilibrium conditions.

$$\boxed{\mathbf{f} : \{0, 1, 2, 3\} \times \mathbb{R}^{60} \rightarrow \mathbb{R}^{59 \cdot 2}} : \mathbf{f} \left( \begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix} \right) = \mathbf{f} \left( \begin{bmatrix} z_t \\ 0_t^1 \\ k_t^2 \\ \dots \\ k_t^{59} \\ k_t^{60} \end{bmatrix} \right) = \begin{bmatrix} a_t^1 \\ \dots \\ a_t^{59} \\ \lambda_t^1 \\ \dots \\ \lambda_t^{59} \end{bmatrix} \left. \begin{array}{l} \text{capital} \\ \text{investment funct.} \\ \text{kkt-multiplier} \\ \text{borrowing contr.} \end{array} \right\}$$

such that:  $\forall h = 1, \dots, 59 :$

$$\left. \begin{array}{l} 0 = \beta \mathbf{E}_t \left[ \frac{R_{t+1} u'(c_{t+1}^{h+1}) + \lambda_t^h}{u'(c_t^h)} \right] - 1 \\ 0 = \lambda_t^h a_t^h \\ 0 \leq \lambda_t^h \\ 0 \leq a_t^h \end{array} \right\} =: \mathbf{G} \left( \begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}, \mathbf{f} \left( \begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix} \right) \right)_h$$

$$\begin{aligned} c_t^h &= k_t^h R_t + l_t^h w_t - a_t^h \\ R_t &= \xi_t \alpha K_t^{\alpha-1} L_t^{1-\alpha} + (1 - \delta_t) \\ w_t &= \xi_t (1 - \alpha) K_t^\alpha L_t^{-\alpha} \\ K_t &= \sum_{h=1}^{60} k_t^h \\ L_t &= \sum_{h=1}^{60} l_t^h \end{aligned}$$



# Deep Equilibrium Net – Architecture

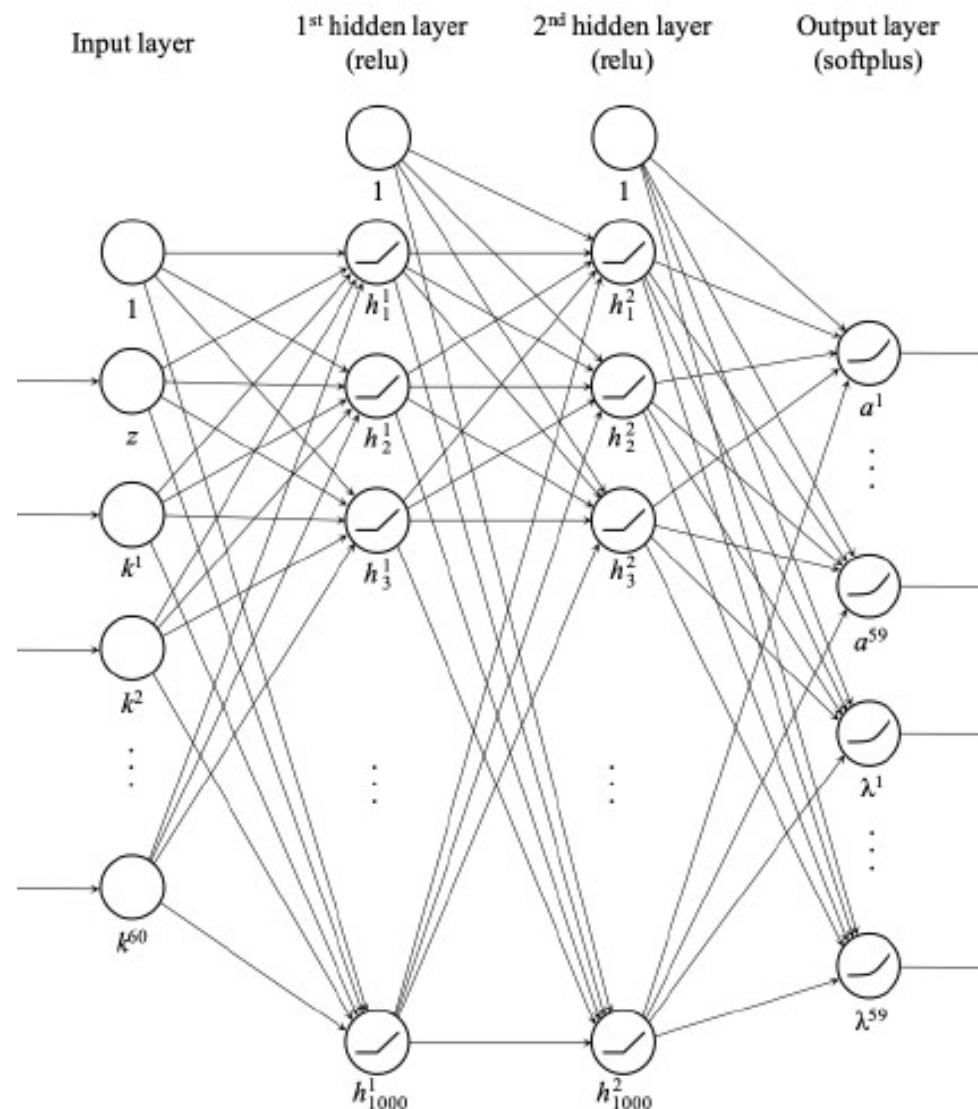
$$\mathcal{N}_\rho : \{0, 1, 2, 3\} \times \mathbb{R}^{60} \rightarrow \mathbb{R}^{59 \cdot 2} :$$

$$\mathcal{N}_\rho \left( \begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix} \right) = \begin{bmatrix} a_t^1 \\ \vdots \\ a_t^{59} \\ \lambda_t^1 \\ \vdots \\ \lambda_t^{59} \end{bmatrix}$$

such that

$$\mathbf{G} \left( \begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}, \mathcal{N}_\rho \left( \begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix} \right) \right) \approx \mathbf{0}$$

$$\hat{\mathbf{x}}_+ = \begin{bmatrix} z_+ \\ 0 \\ \hat{a}^{[1:N-1]}(\mathbf{x}) \end{bmatrix}$$

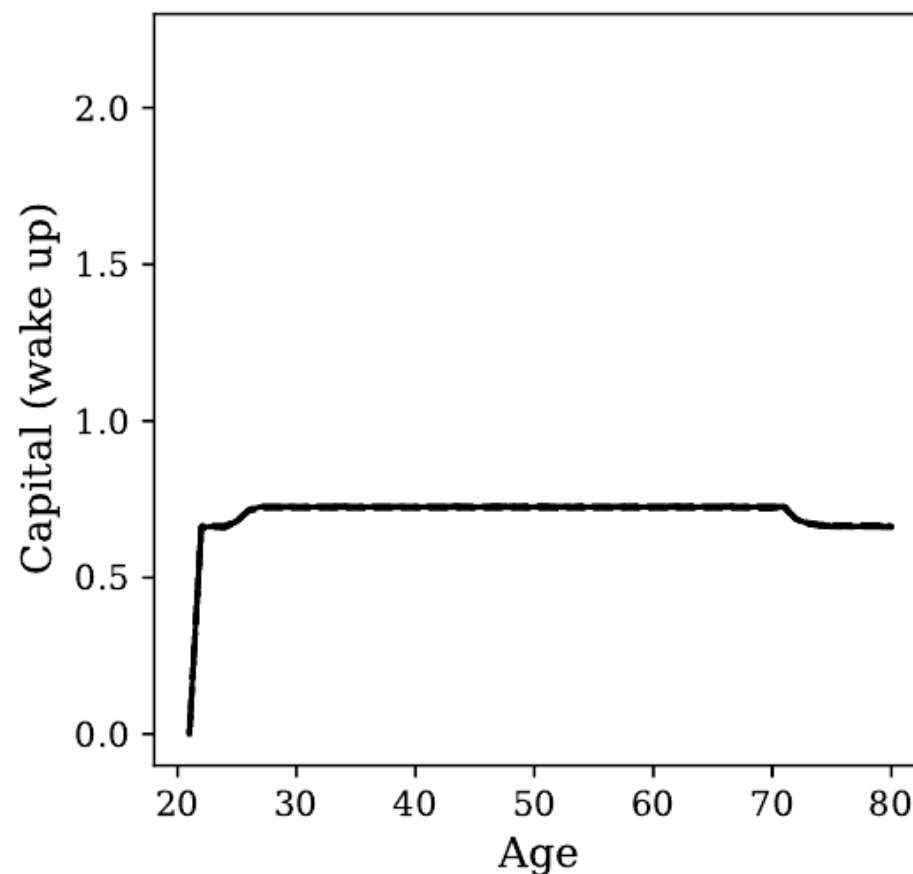
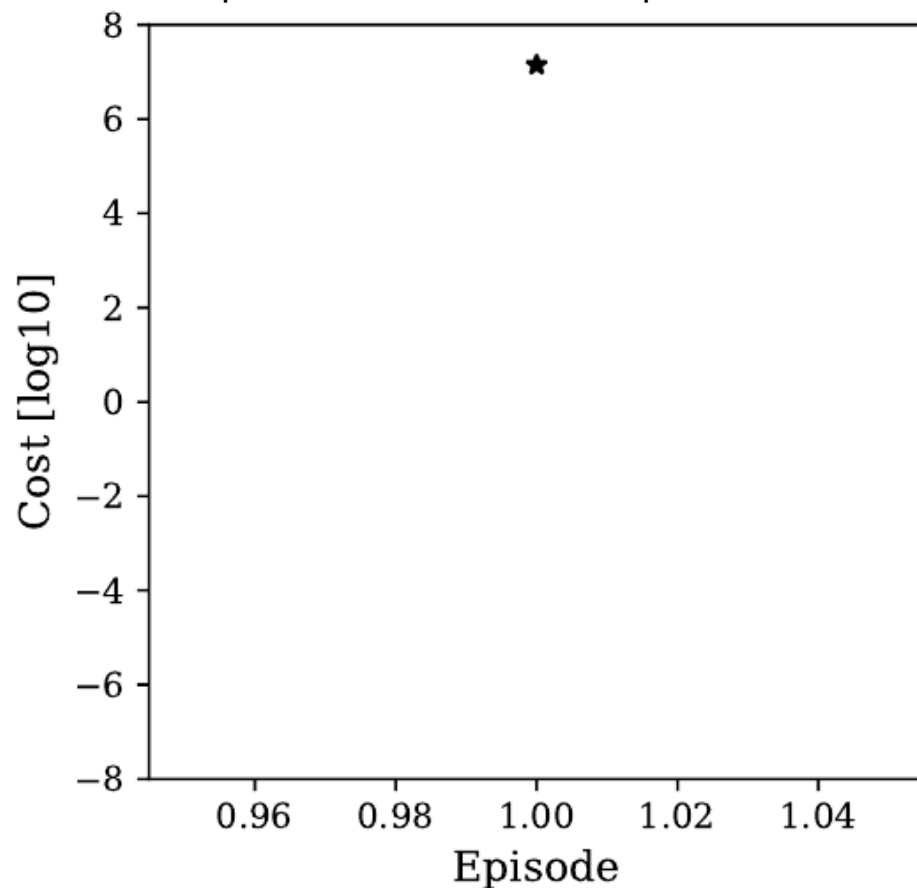


# Learning the Equilibrium

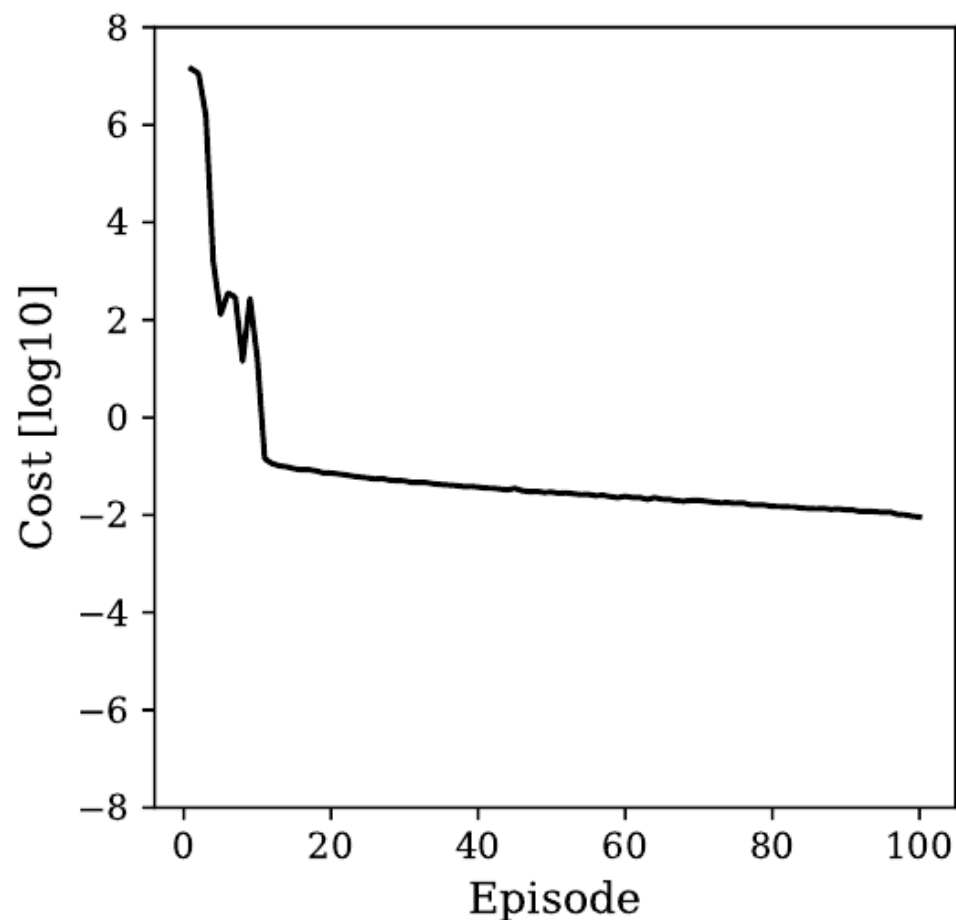
Def.: **Episode**: the set of  $T$  simulated periods.

**Epoch**: when the whole dataset is passed through the algorithm.

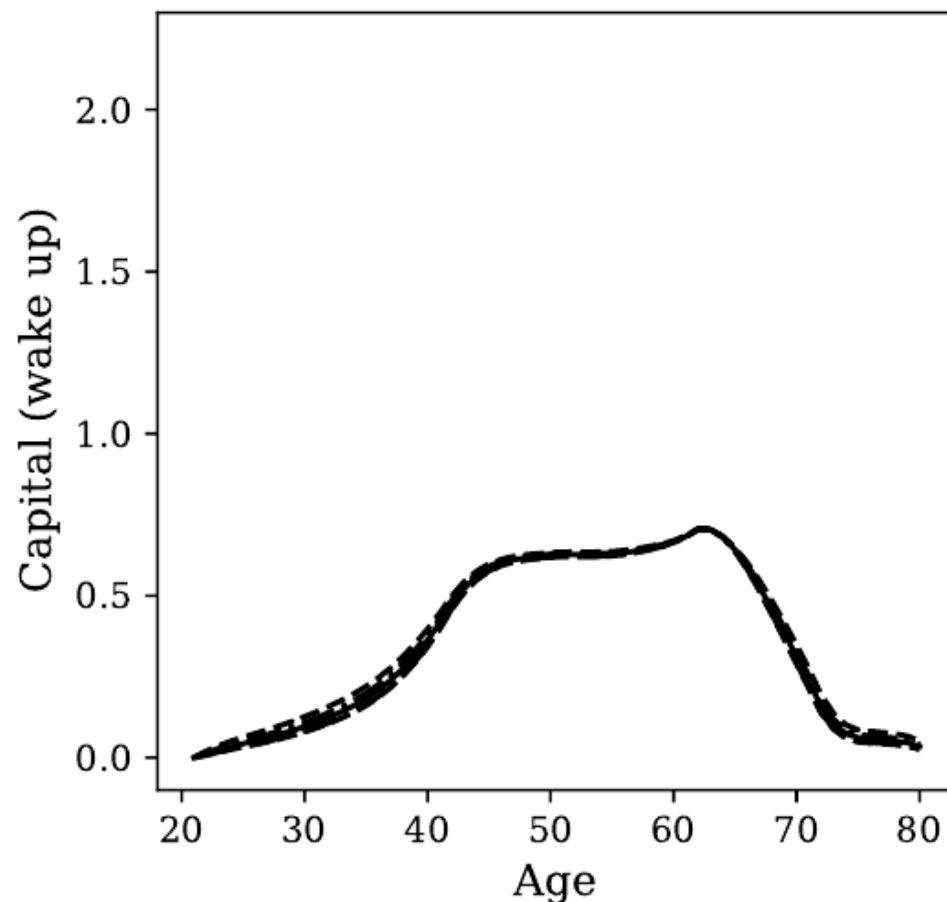
Per epoch, the neural network parameters are updated  $T/m$  times.



# Learning the Equilibrium



Typical runtime: 2.1 [sec/Episode]

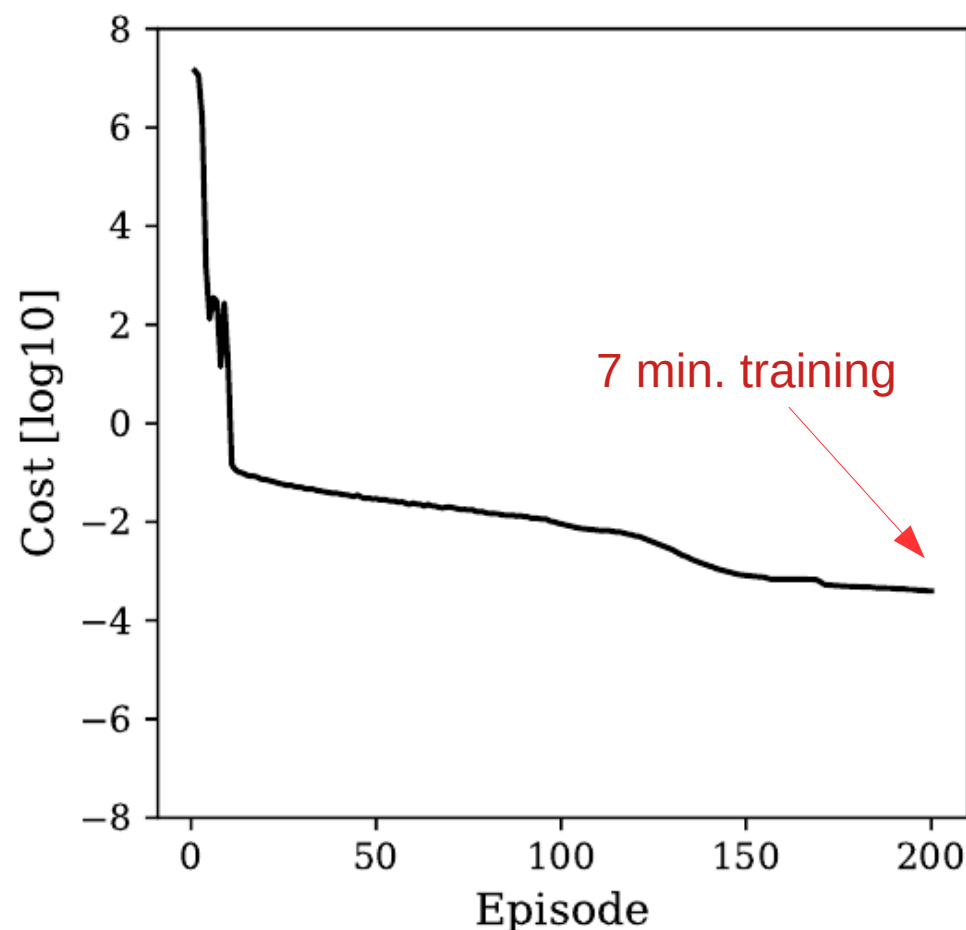


Learning rate:  $10^{-5}$

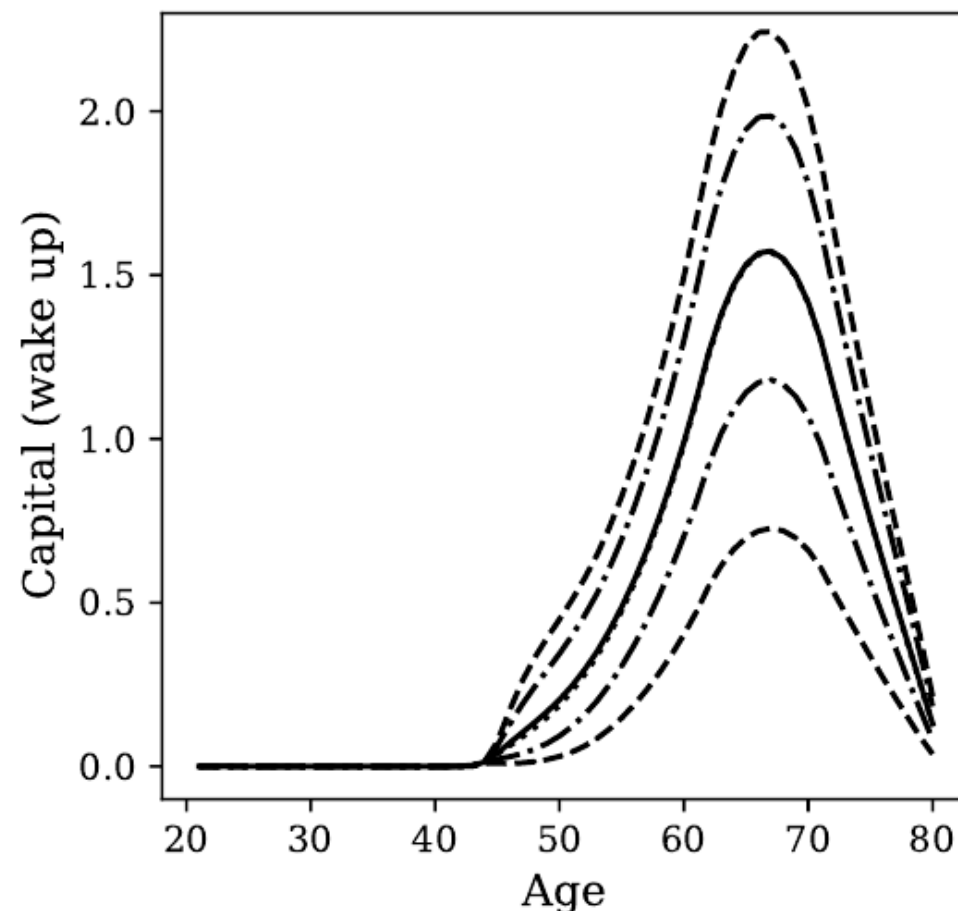
Batch size: 1,000; 10,000 periods/Episode

# Learning the Equilibrium

Solid line: **mean** over 10,000 simulated periods  
Dash-dotted lines show the **10th and 90th percentile**  
Dashed lines show the **0.1th and 99.9th percentile**.



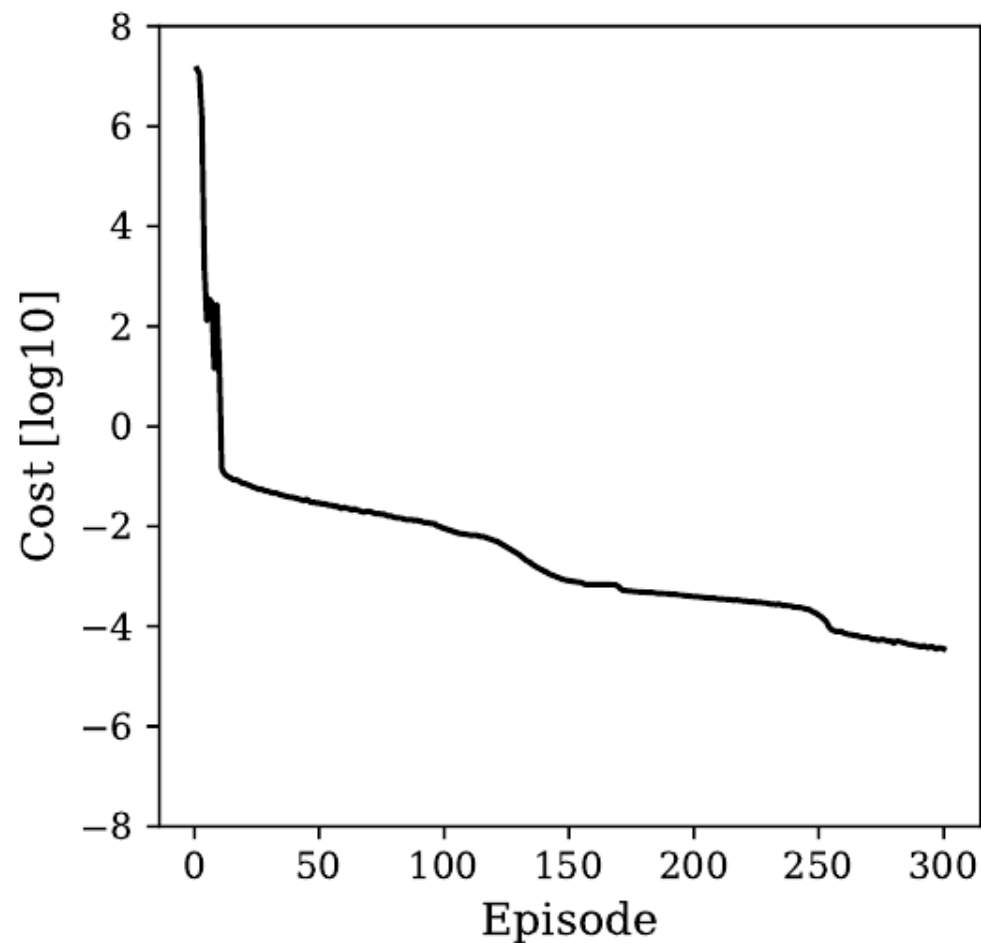
Typical runtime: 2.1 [sec/Episode]



Learning rate:  $10^{-5}$

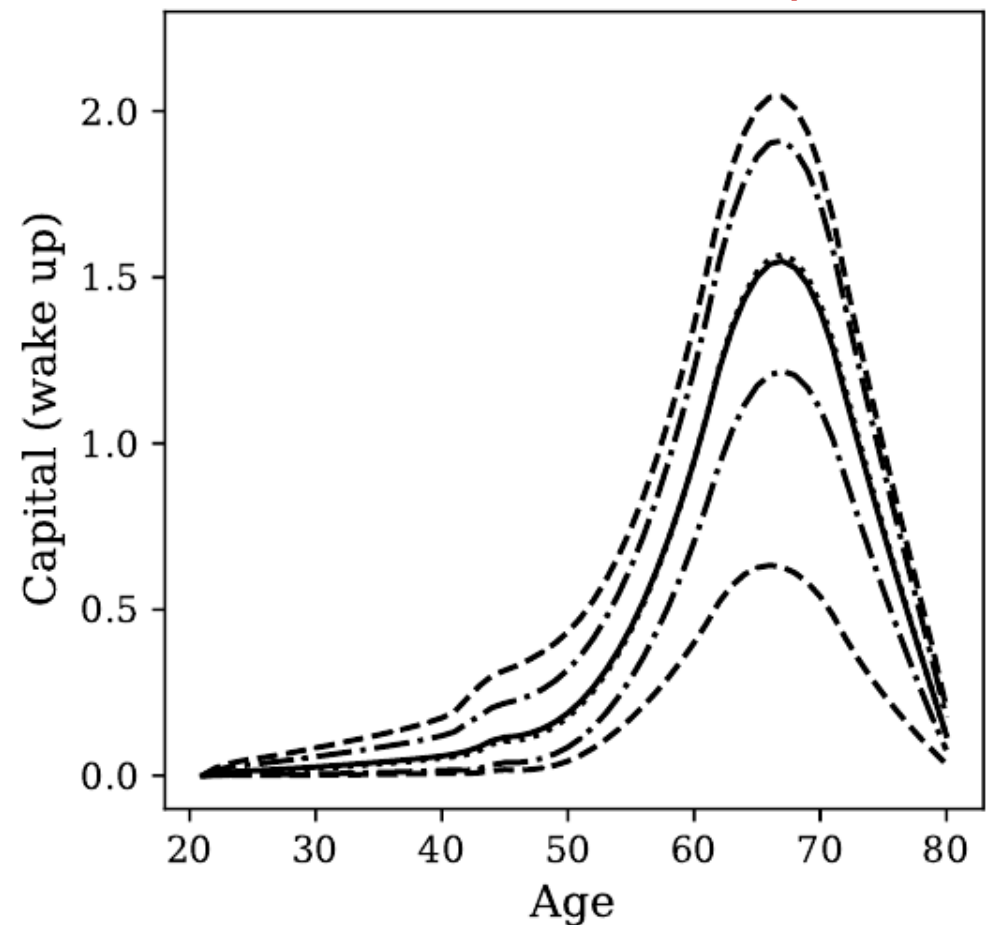
Batch size: 1,000; 10,000 periods/Episode

# Learning the Equilibrium



Typical runtime: 2.1 [sec/Episode]

Solid line: **mean** over 10,000 simulated periods  
Dash-dotted lines show the **10th and 90th percentile**  
Dashed lines show the **0.1th and 99.9th percentile**.

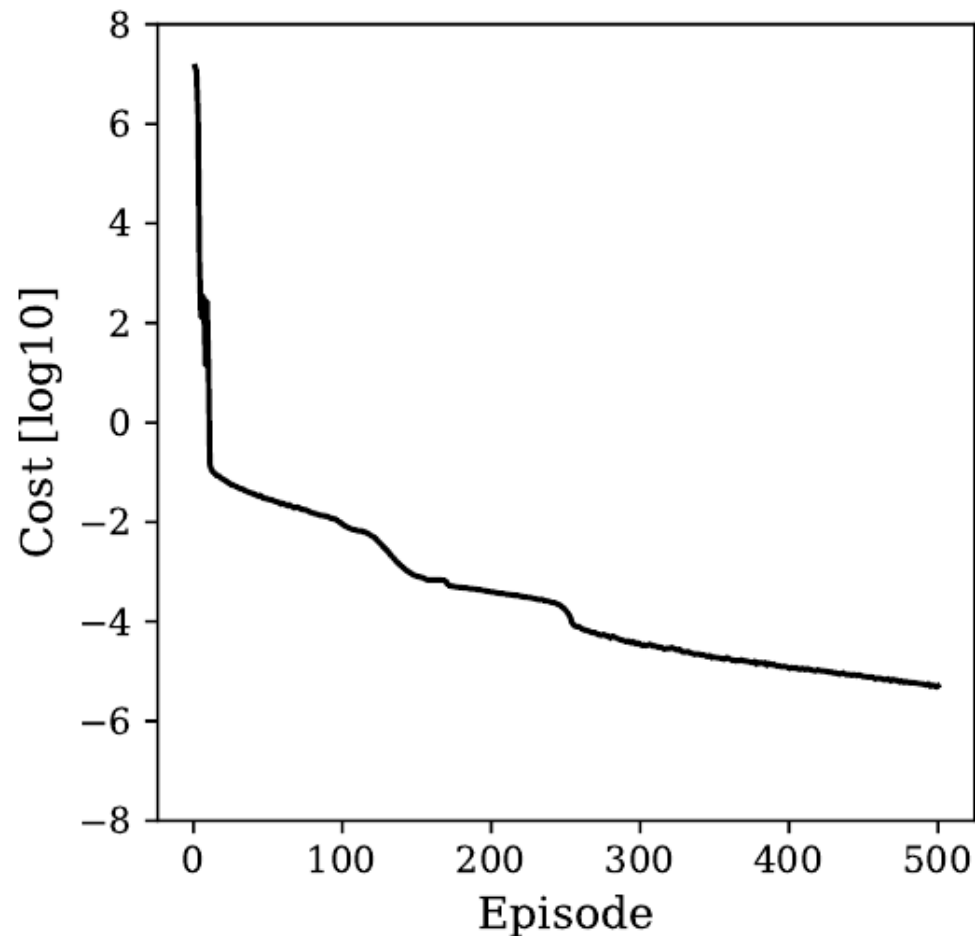


Learning rate:  $10^{-5}$

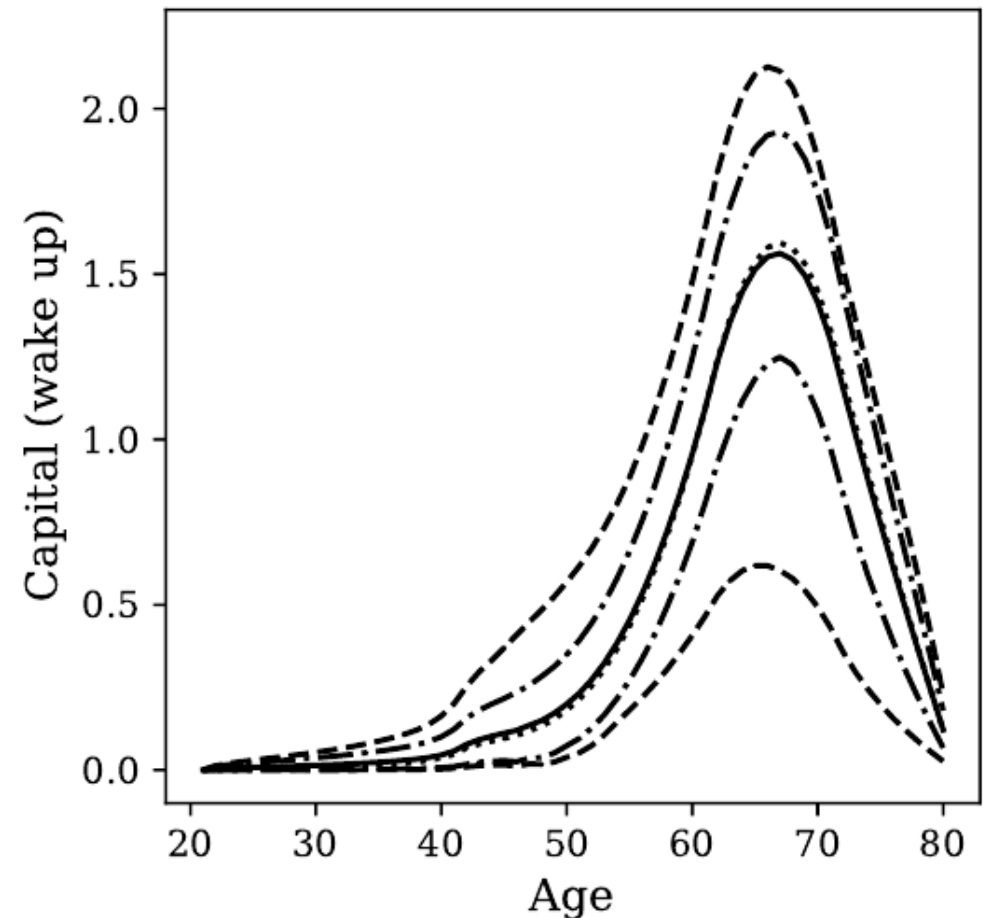
Batch size: 1,000; 10,000 periods/Episode

# Learning the Equilibrium

Solid line: **mean** over 10,000 simulated periods  
Dash-dotted lines show the **10th and 90th percentile**  
Dashed lines show the **0.1th and 99.9th percentile**.



Typical runtime: 2.1 [sec/Episode]

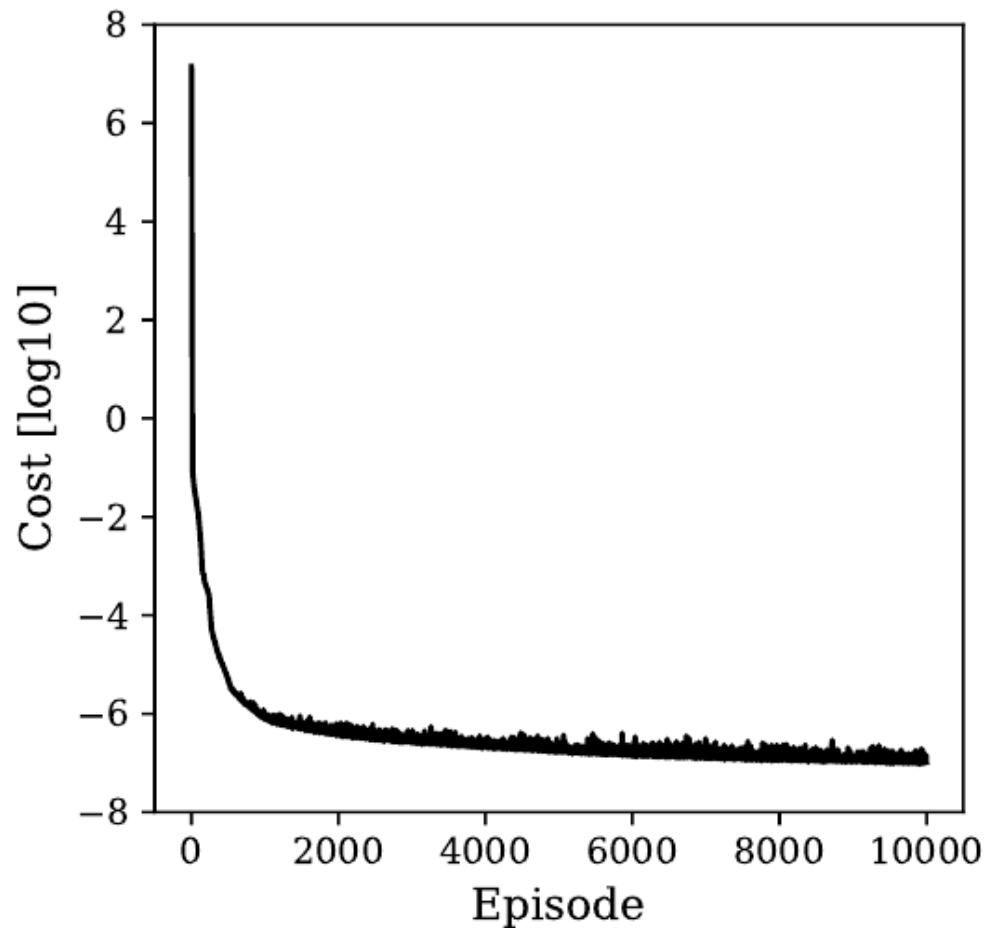


Learning rate:  $10^{-5}$

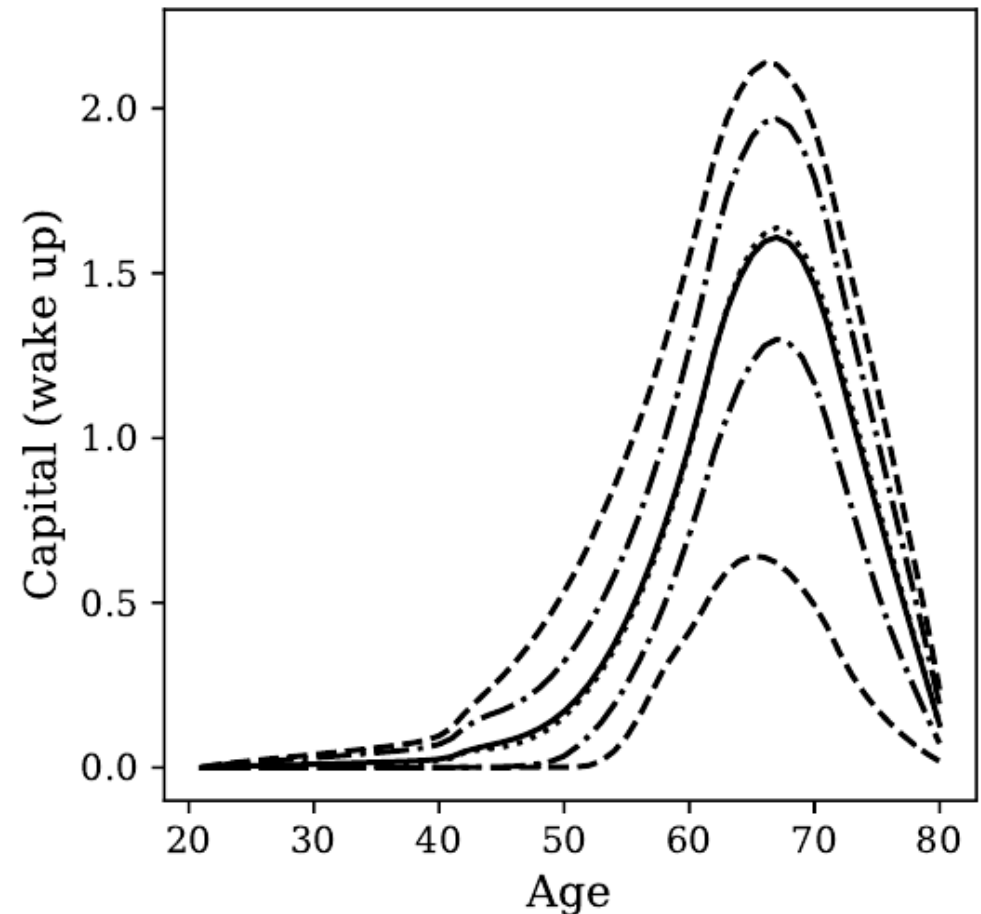
Batch size: 1,000; 10,000 periods/Episode

# Learning the Equilibrium

Solid line: **mean** over 10,000 simulated periods  
Dash-dotted lines show the **10th and 90th percentile**  
Dashed lines show the **0.1th and 99.9th percentile**.



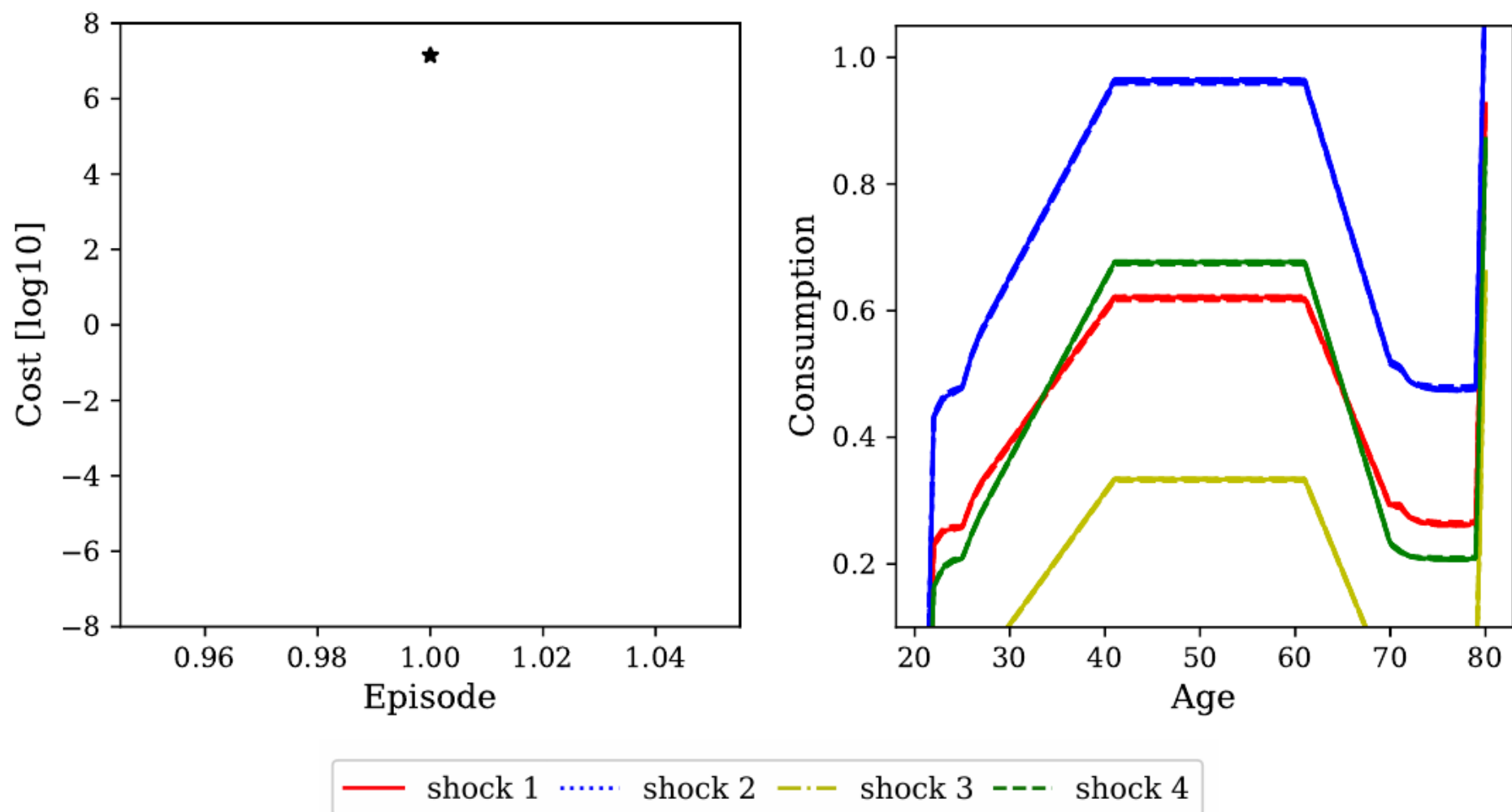
Typical runtime: 2.1 [sec/Episode]



Learning rate:  $10^{-5}$

Batch size: 1,000; 10,000 periods/Episode

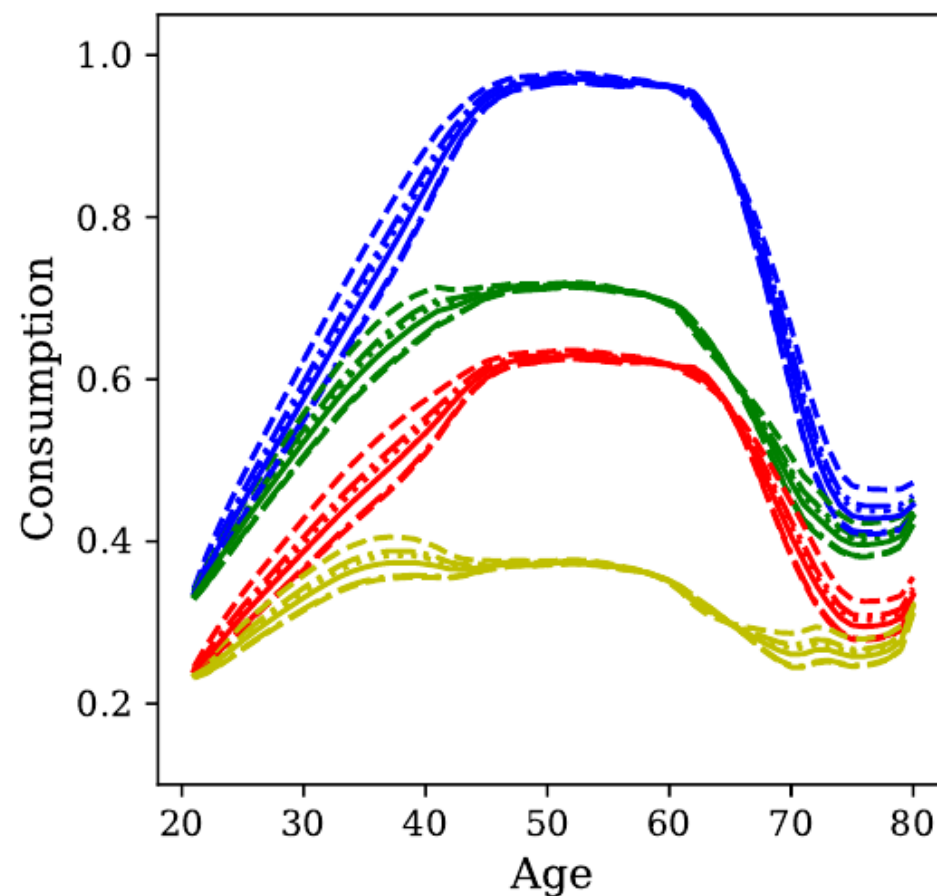
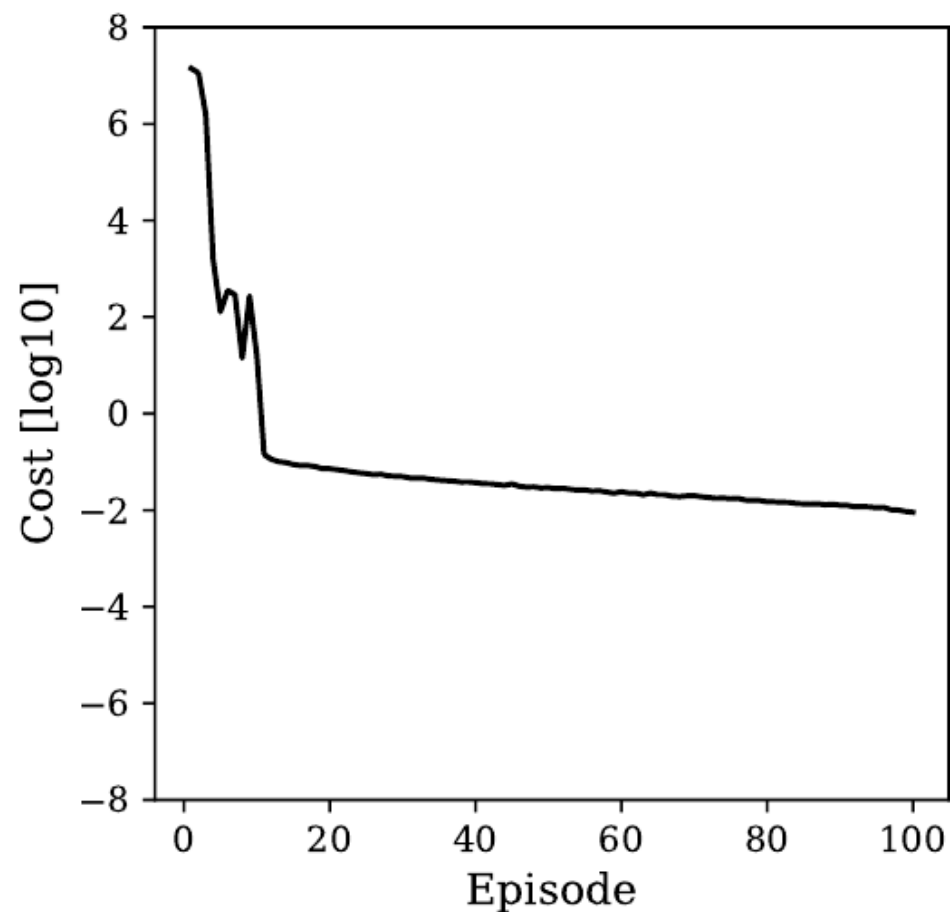
# Learning the Equilibrium



shock 1:  $\delta = 0.5$ ,  $\xi = 0.85$ , shock 2:  $\delta = 0.5$ ,  $\xi = 1.15$ , shock 3:  $\delta = 0.9$ ,  $\xi = 0.85$ , shock 4:  $\delta = 0.9$ ,  $\xi = 1.15$



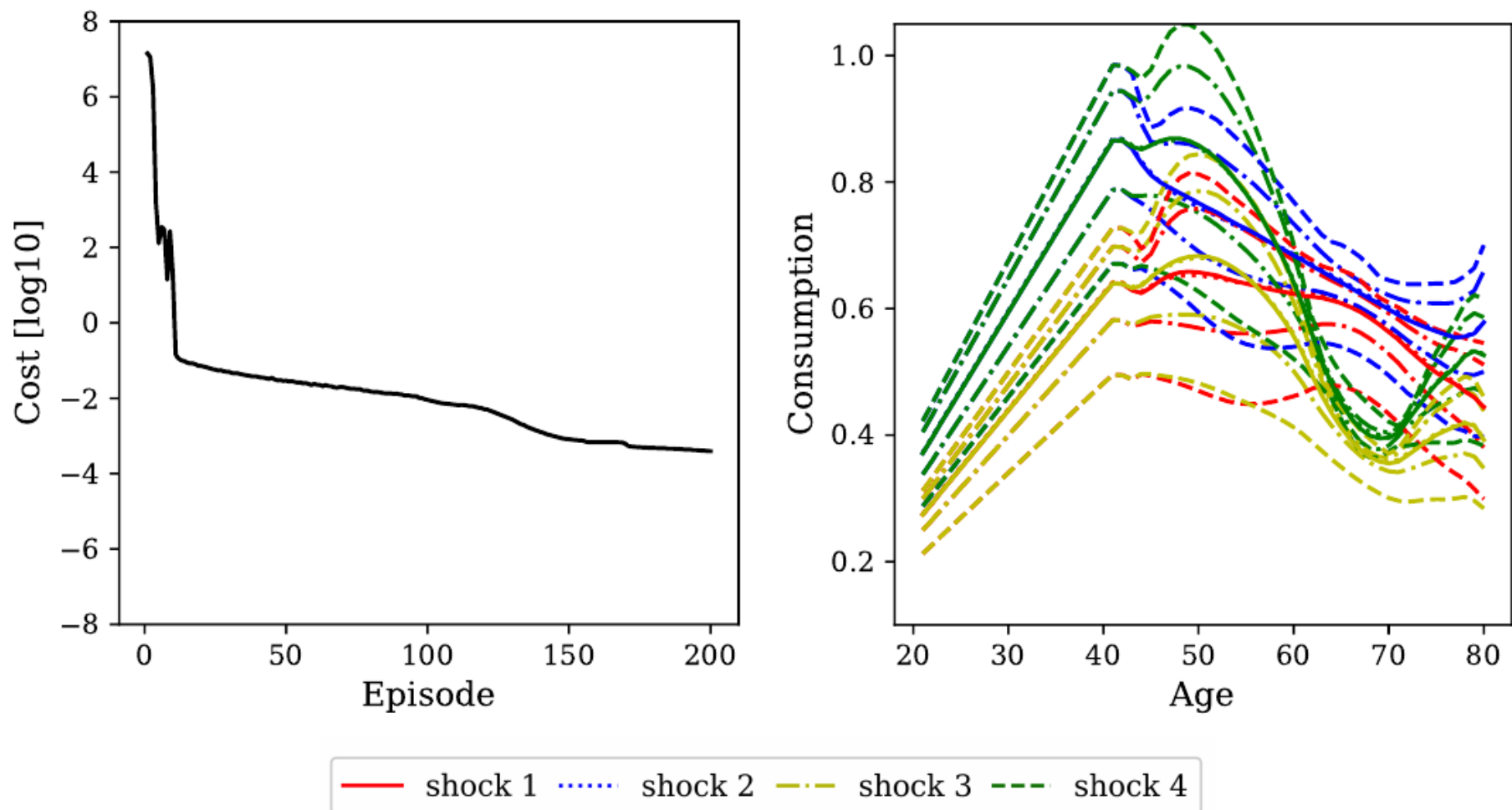
# Learning the Equilibrium



— shock 1    ..... shock 2    -.-.- shock 3    --- shock 4

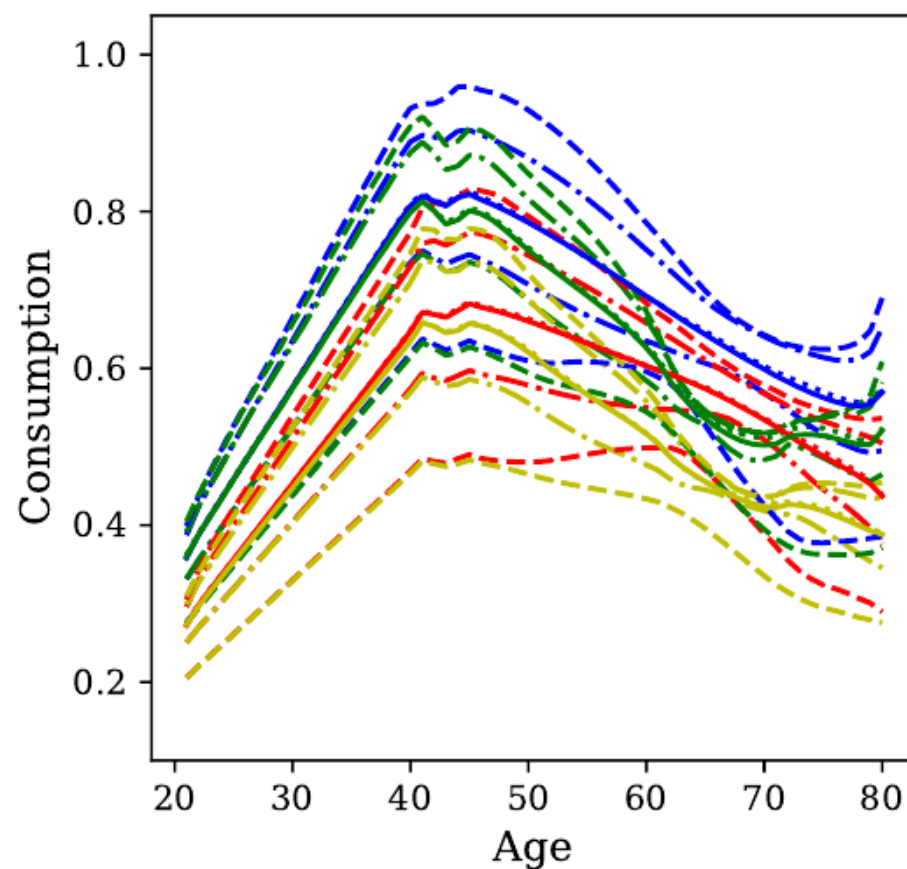
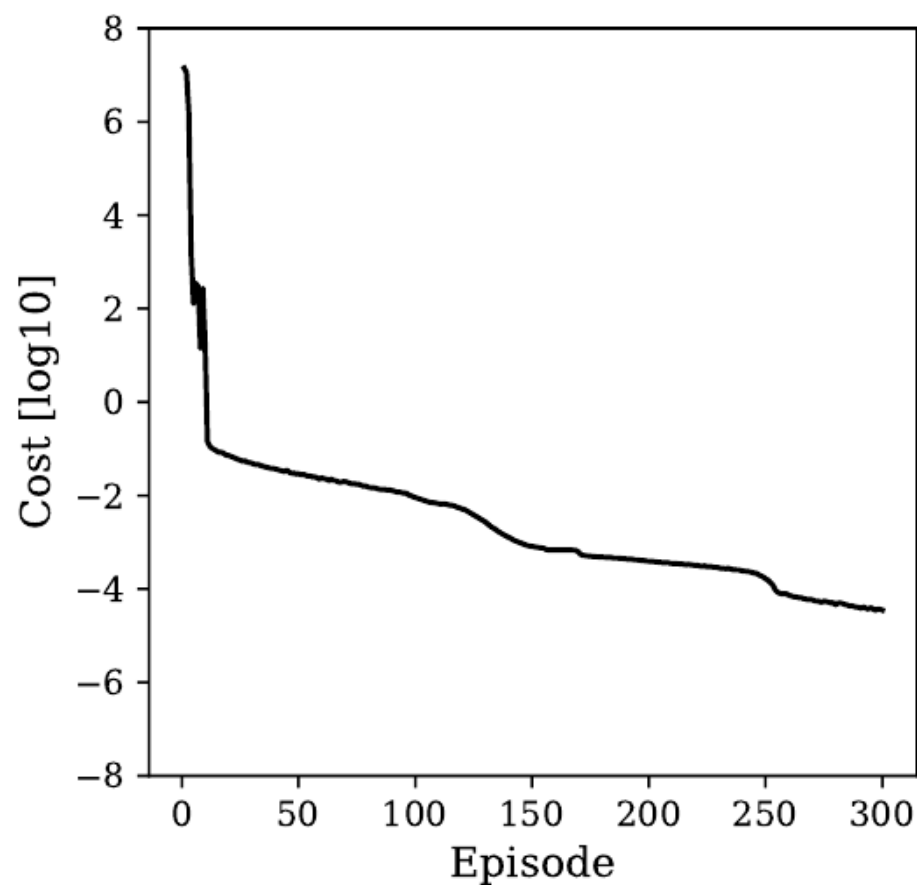
shock 1:  $\delta = 0.5$ ,  $\xi = 0.85$ , shock 2:  $\delta = 0.5$ ,  $\xi = 1.15$ , shock 3:  $\delta = 0.9$ ,  $\xi = 0.85$ , shock 4:  $\delta = 0.9$ ,  $\xi = 1.15$

# Learning the Equilibrium



shock 1:  $\delta = 0.5$ ,  $\xi = 0.85$ , shock 2:  $\delta = 0.5$ ,  $\xi = 1.15$ , shock 3:  $\delta = 0.9$ ,  $\xi = 0.85$ , shock 4:  $\delta = 0.9$ ,  $\xi = 1.15$

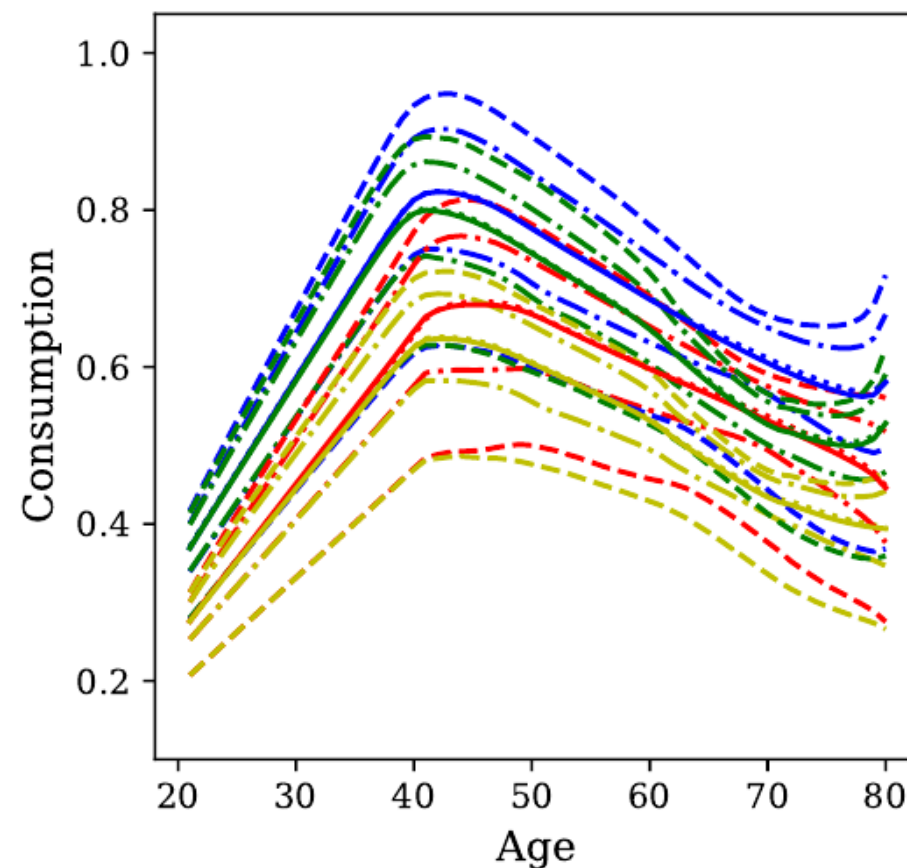
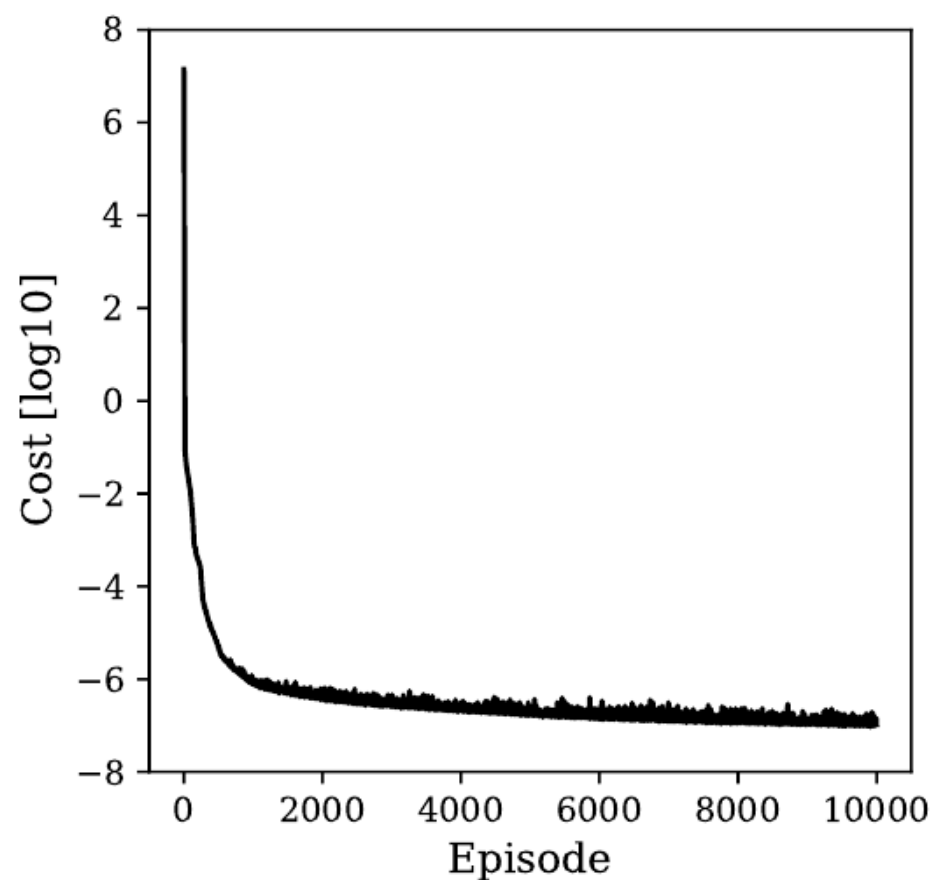
# Learning the Equilibrium



— shock 1    ..... shock 2    -.-.- shock 3    --- shock 4

shock 1:  $\delta = 0.5$ ,  $\xi = 0.85$ , shock 2:  $\delta = 0.5$ ,  $\xi = 1.15$ , shock 3:  $\delta = 0.9$ ,  $\xi = 0.85$ , shock 4:  $\delta = 0.9$ ,  $\xi = 1.15$

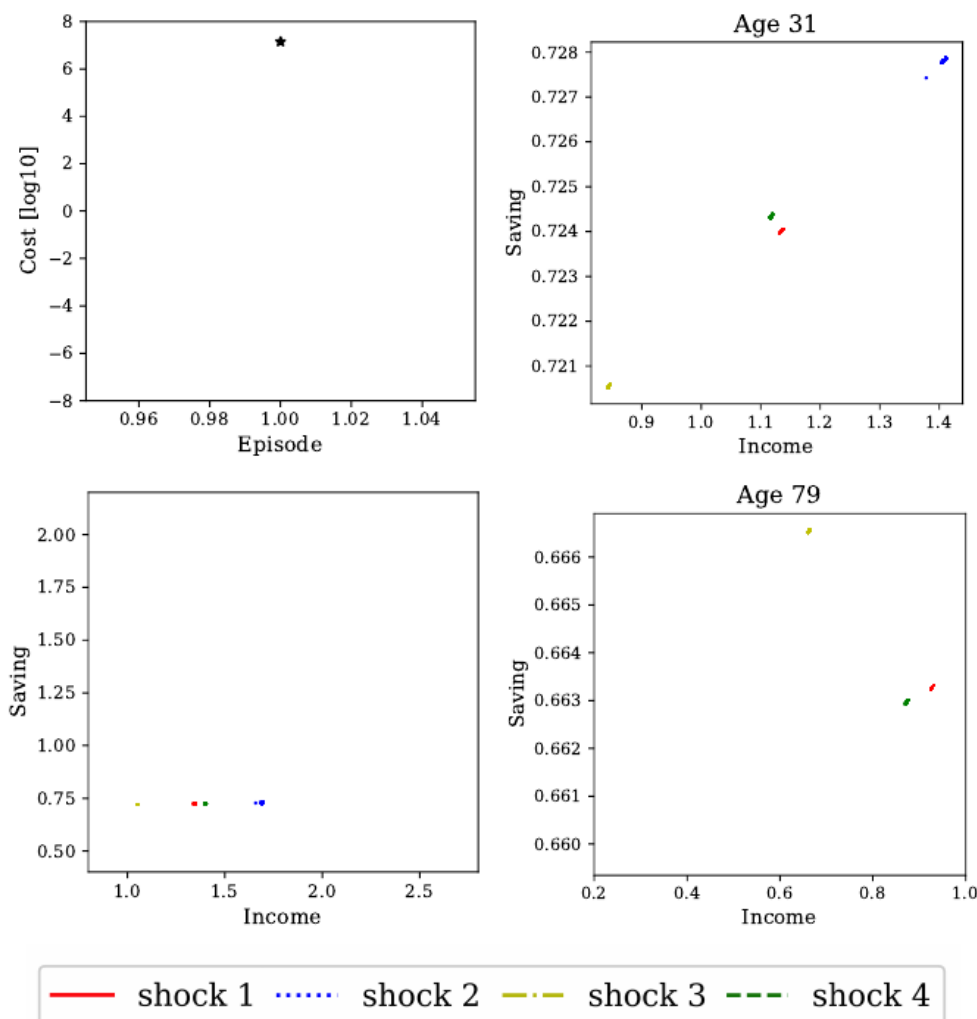
# Learning the Equilibrium



— shock 1    ..... shock 2    -.-.- shock 3    --- shock 4

shock 1:  $\delta = 0.5$ ,  $\xi = 0.85$ , shock 2:  $\delta = 0.5$ ,  $\xi = 1.15$ , shock 3:  $\delta = 0.9$ ,  $\xi = 0.85$ , shock 4:  $\delta = 0.9$ ,  $\xi = 1.15$

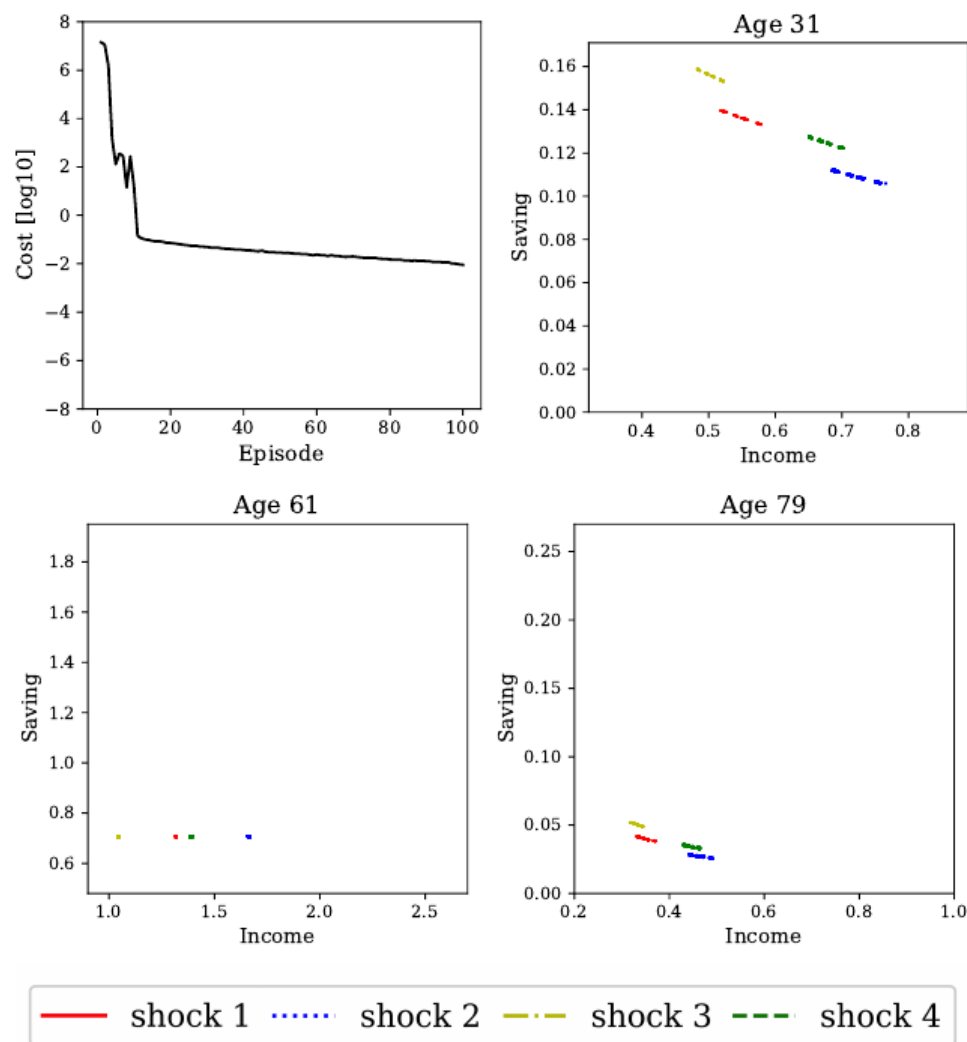
# Learning the Equilibrium



Saving: in capital  
 $w * l + k * r$  : Income  
 $k * r$  : financial wealth  
 $w * l$  : labor income

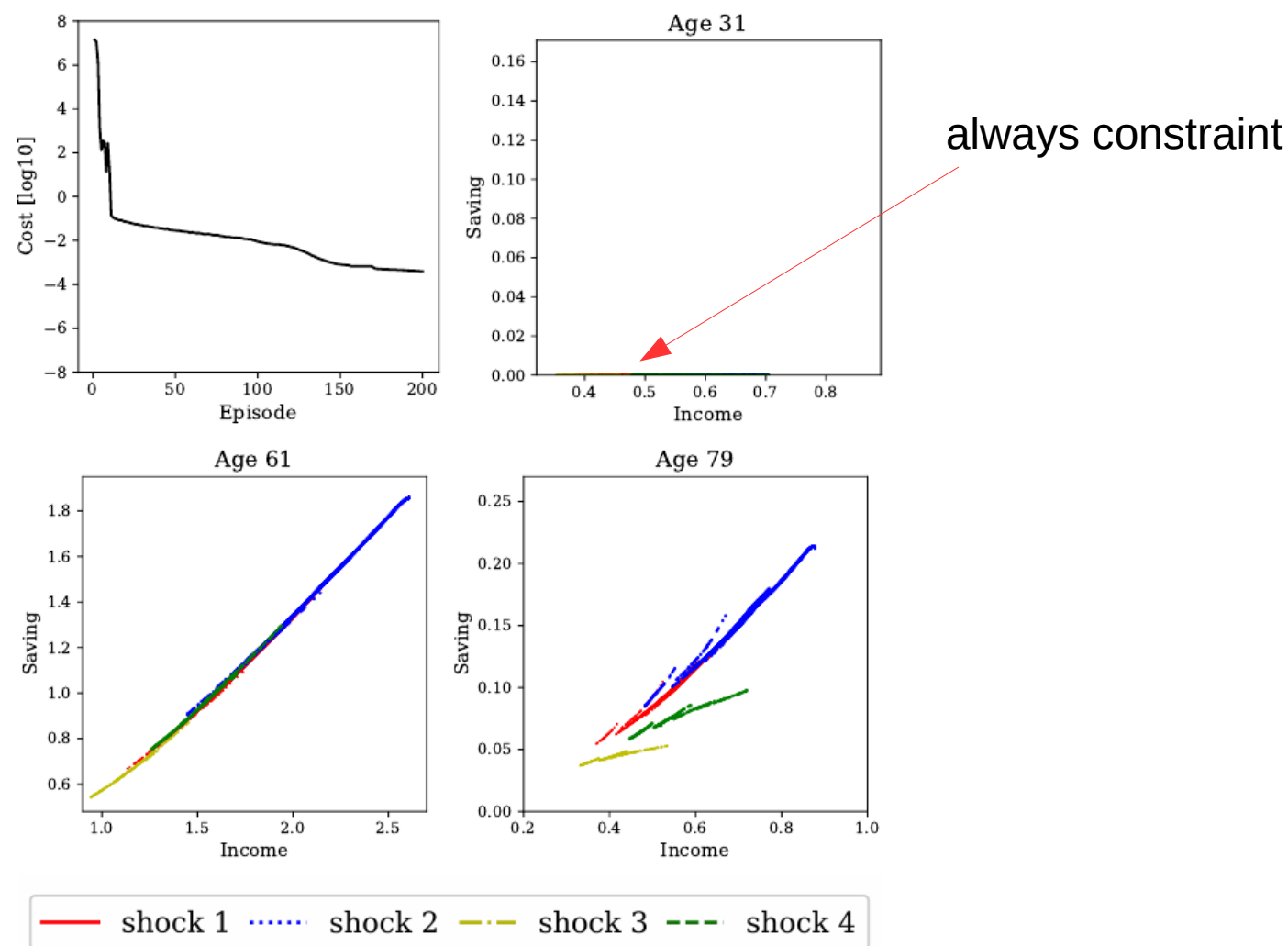
shock 1 :  $\delta = 0.5, \xi = 0.85$ , shock 2 :  $\delta = 0.5, \xi = 1.15$ , shock 3 :  $\delta = 0.9, \xi = 0.85$  shock 4 :  $\delta = 0.9, \xi = 1.15$

# Learning the Equilibrium



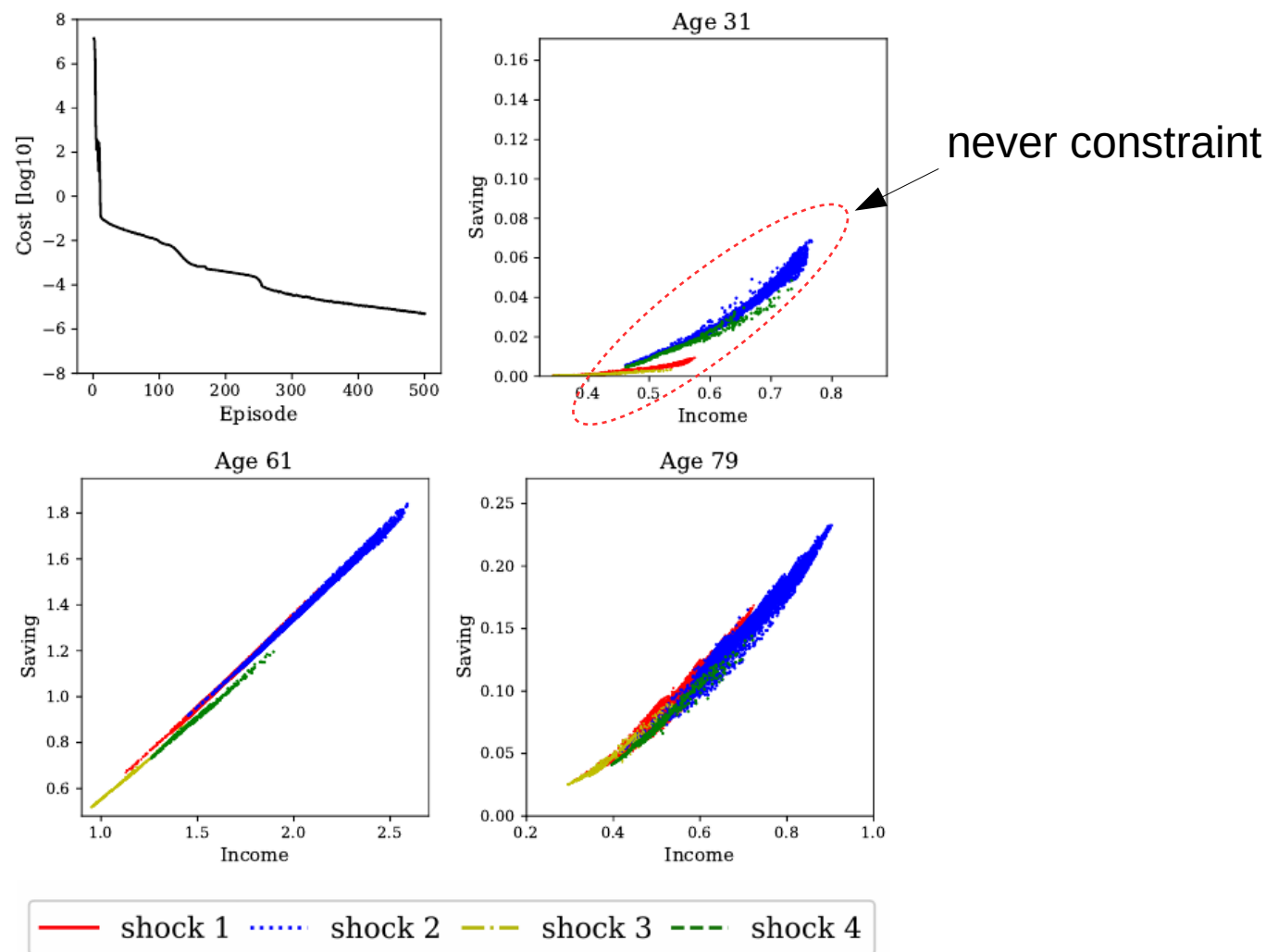
shock 1 :  $\delta = 0.5, \xi = 0.85$ , shock 2 :  $\delta = 0.5, \xi = 1.15$ , shock 3 :  $\delta = 0.9, \xi = 0.85$  shock 4 :  $\delta = 0.9, \xi = 1.15$

# Learning the Equilibrium



shock 1 :  $\delta = 0.5, \xi = 0.85$ , shock 2 :  $\delta = 0.5, \xi = 1.15$ , shock 3 :  $\delta = 0.9, \xi = 0.85$  shock 4 :  $\delta = 0.9, \xi = 1.15$

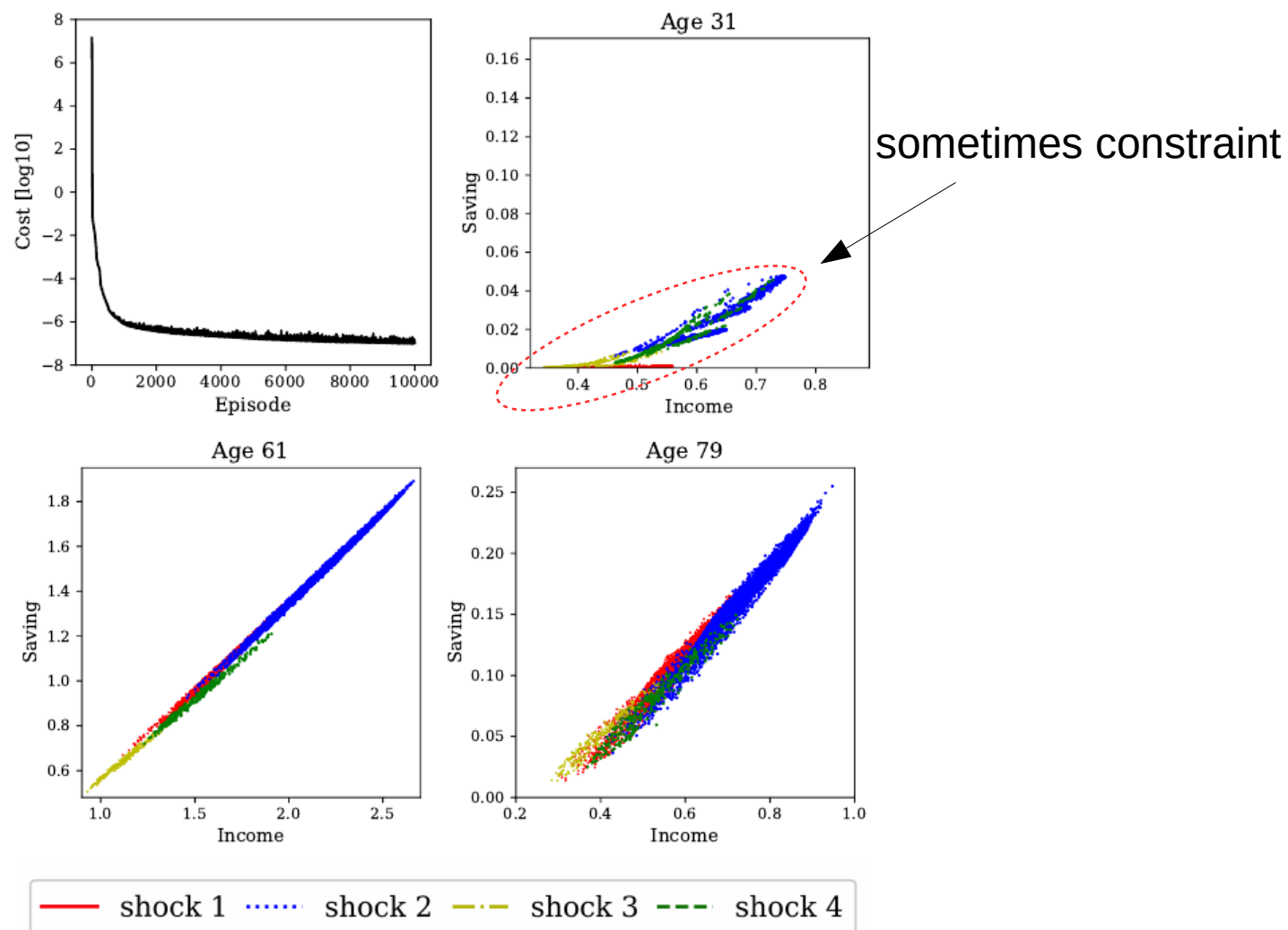
# Learning the Equilibrium



shock 1 :  $\delta = 0.5, \xi = 0.85$ , shock 2 :  $\delta = 0.5, \xi = 1.15$ , shock 3 :  $\delta = 0.9, \xi = 0.85$  shock 4 :  $\delta = 0.9, \xi = 1.15$

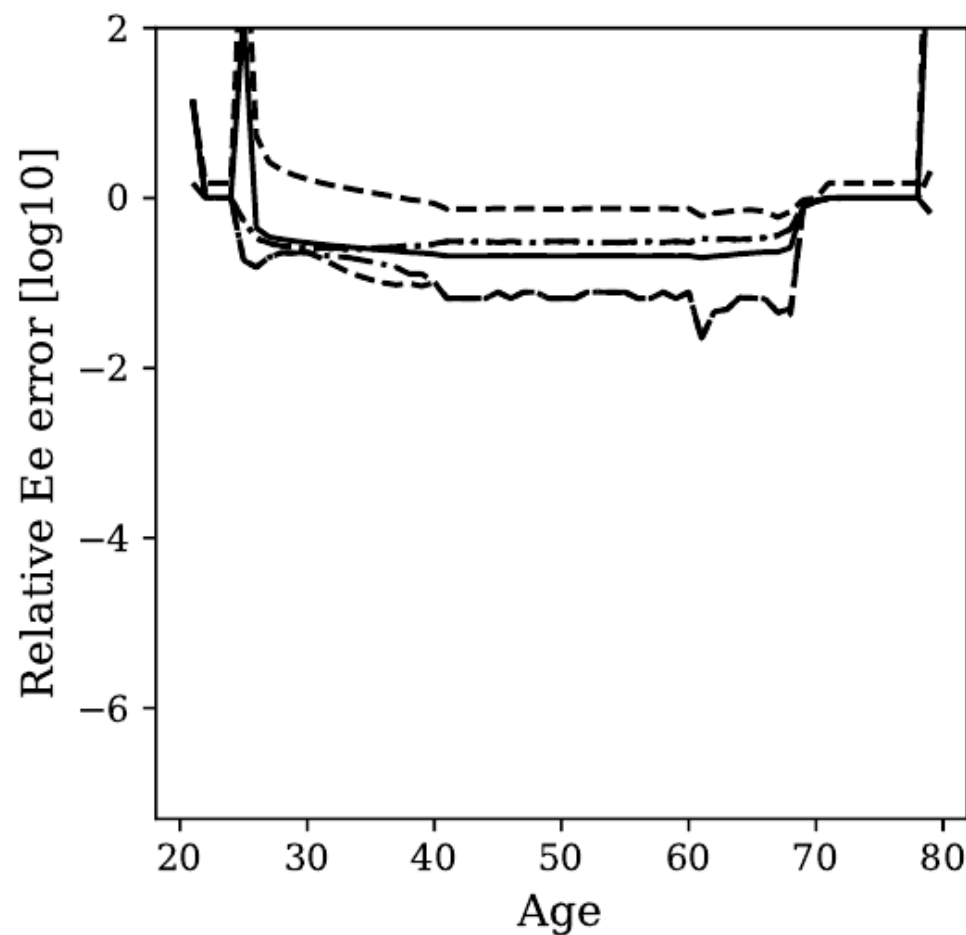
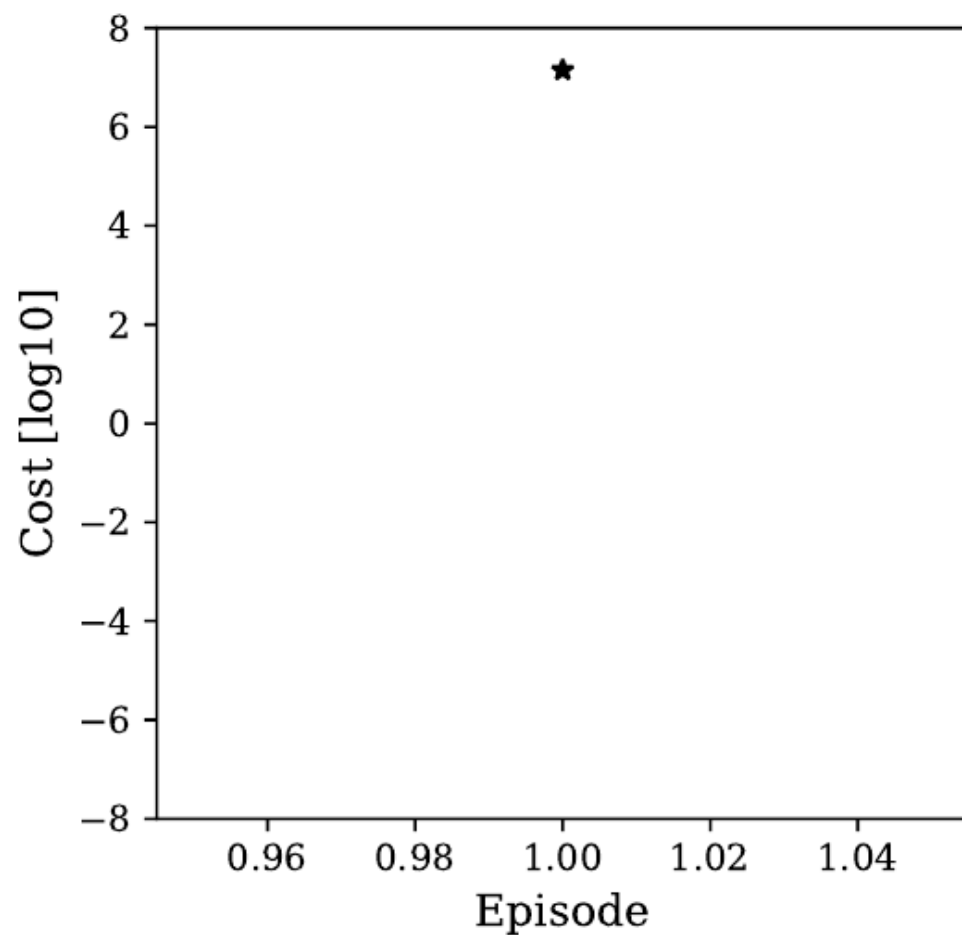


# Learning the Equilibrium

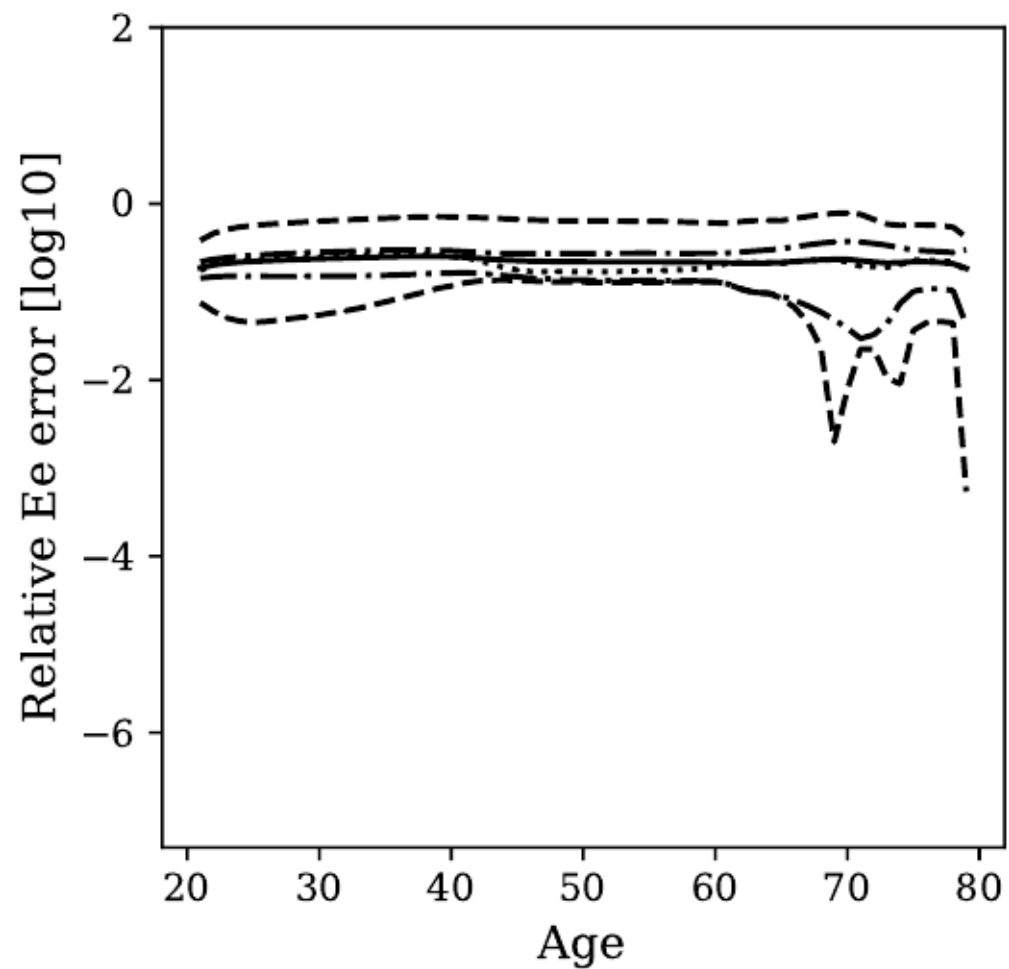
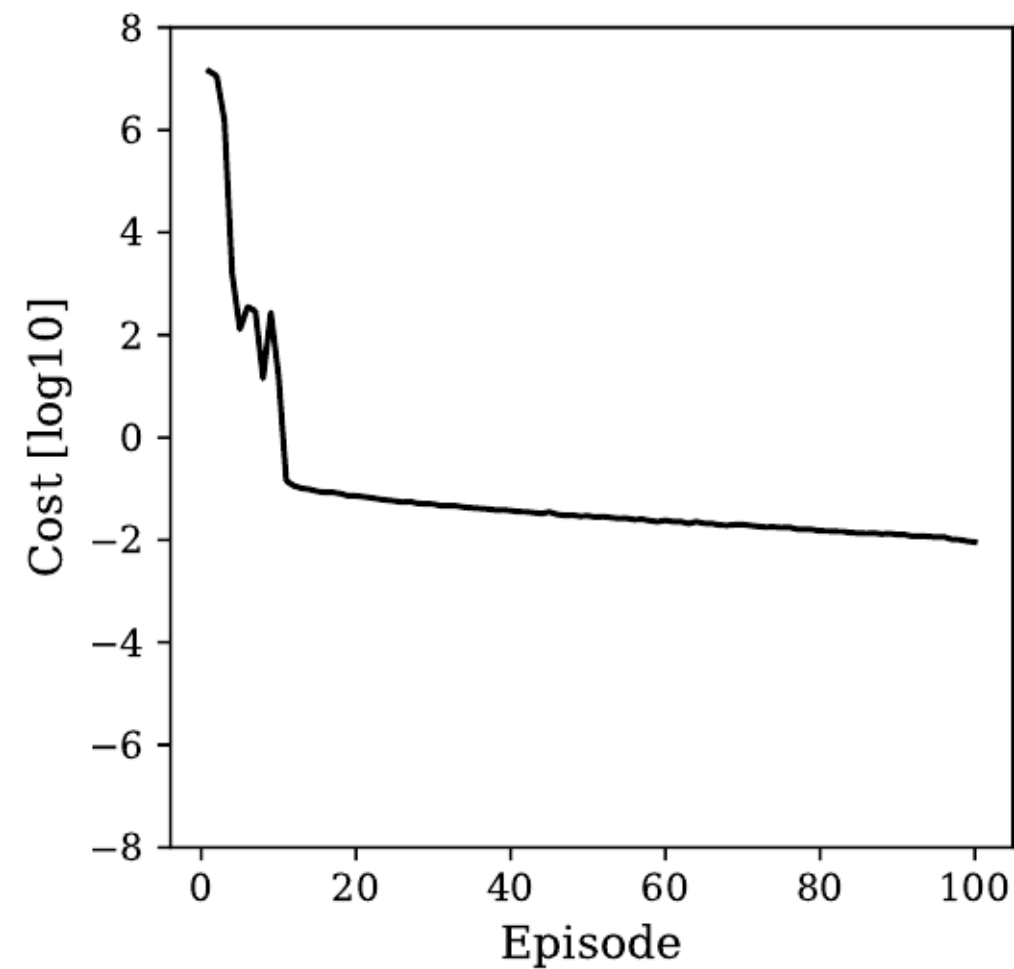


shock 1 :  $\delta = 0.5, \xi = 0.85$ , shock 2 :  $\delta = 0.5, \xi = 1.15$ , shock 3 :  $\delta = 0.9, \xi = 0.85$  shock 4 :  $\delta = 0.9, \xi = 1.15$

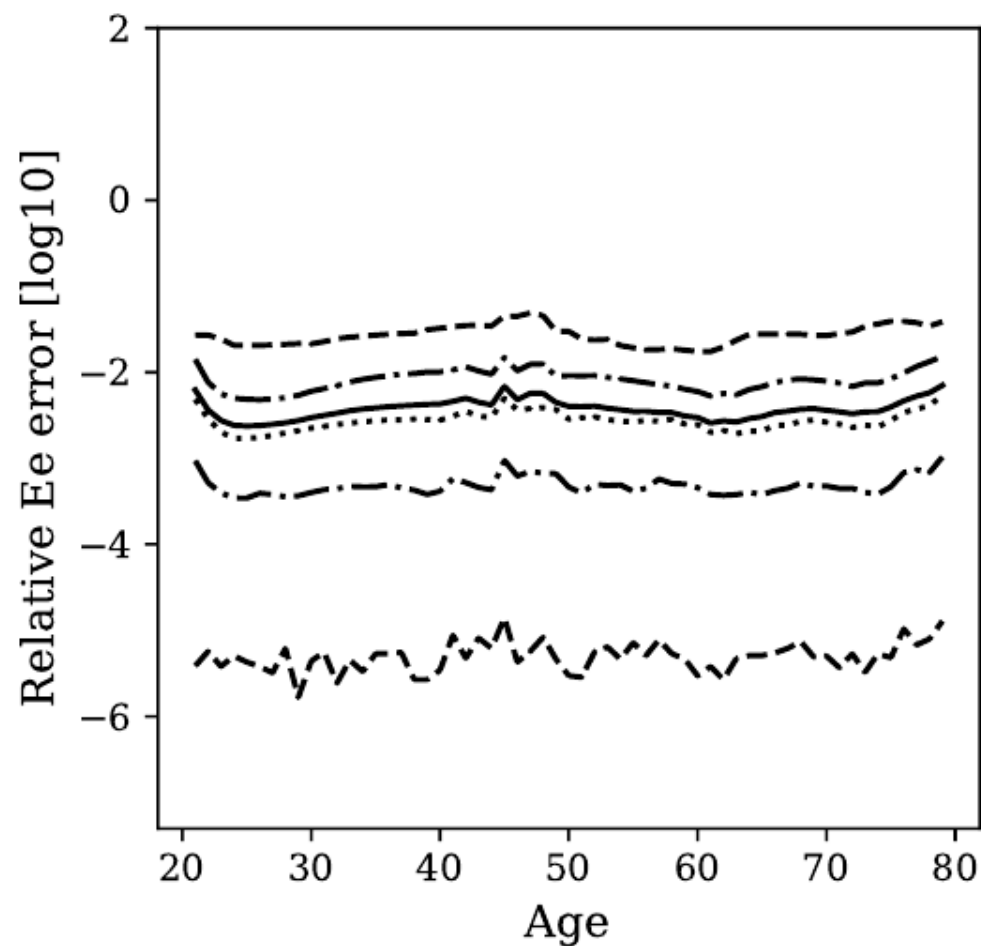
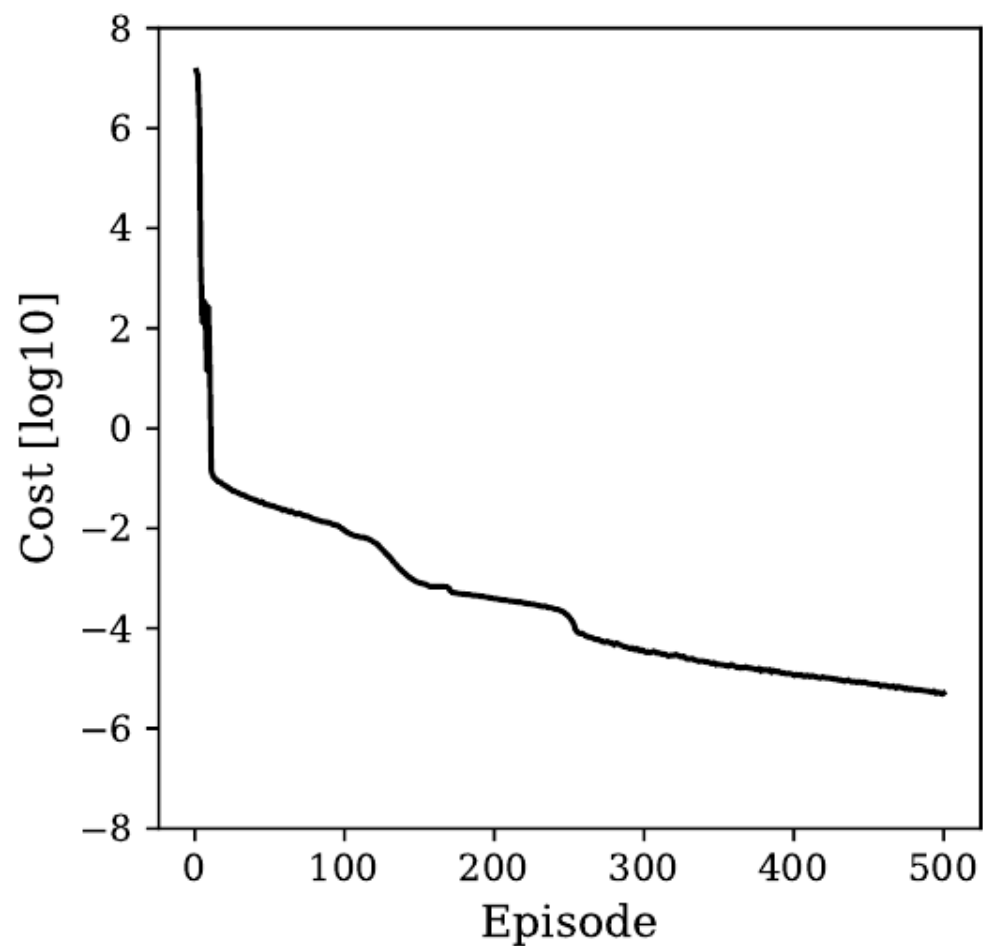
# Learning the Equilibrium



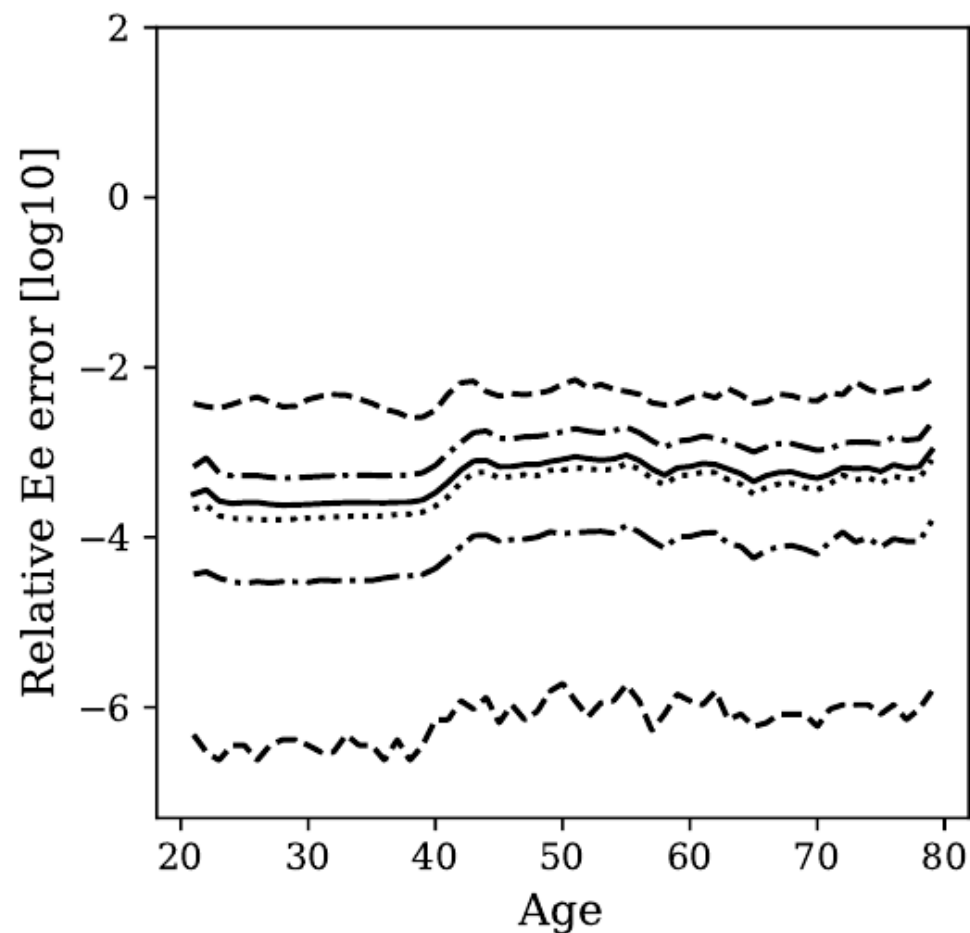
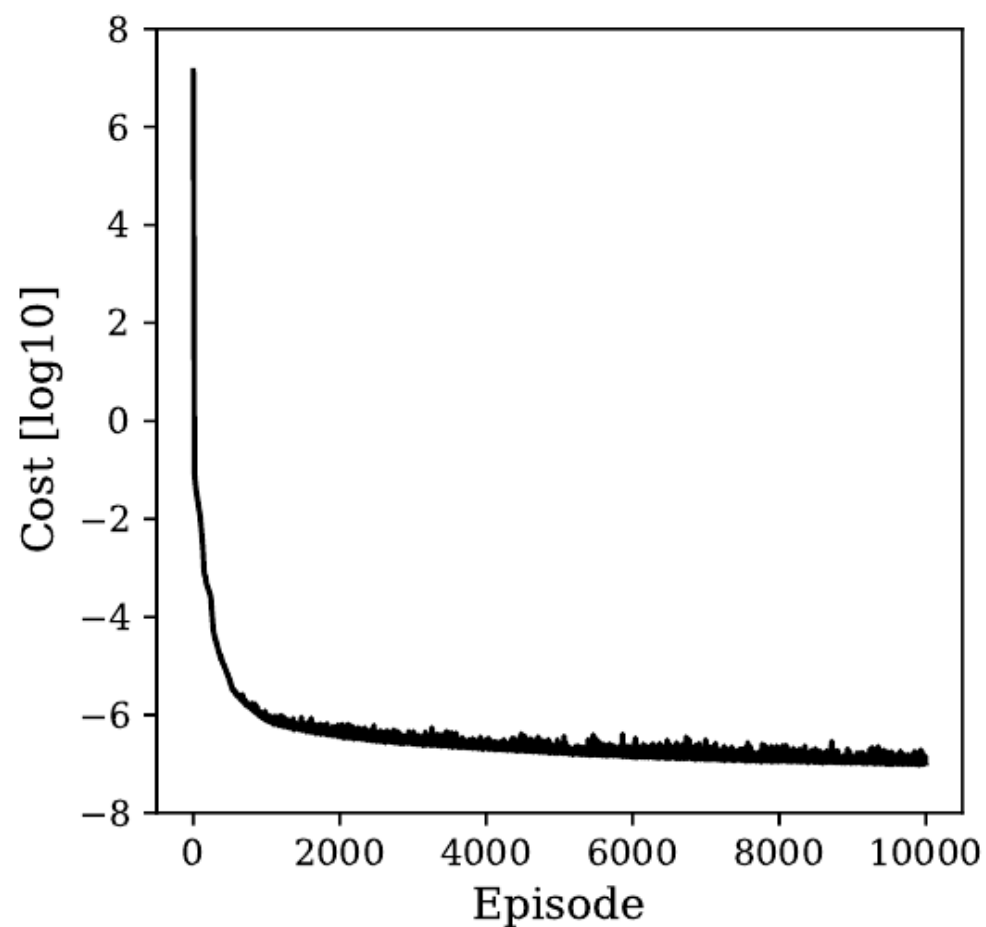
# Learning the Equilibrium



# Learning the Equilibrium



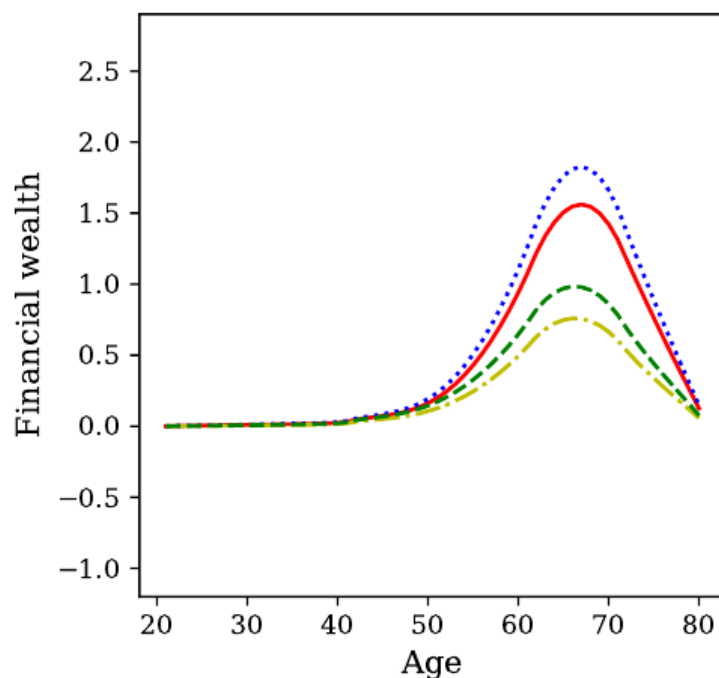
# Learning the Equilibrium



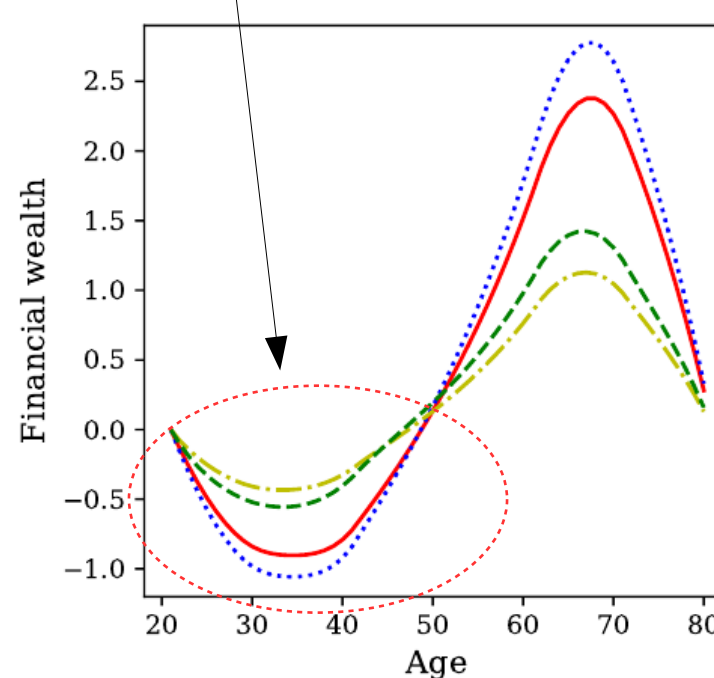
# Loosening the borrowing constraint

**Financial wealth** = value of the **capital saved** in the **last period** after returns realized.

Young agents take up debt, pay back later as their labour endowment increases in their later years.



(a)  $k_t^h \geq 0$



(b)  $k_t^h \geq -1$

— shock 1    ..... shock 2    -.-.- shock 3    ---- shock 4

shock 1 :  $\delta = 0.5, \xi = 0.85$ , shock 2 :  $\delta = 0.5, \xi = 1.15$ , shock 3 :  $\delta = 0.9, \xi = 0.85$  shock 4 :  $\delta = 0.9, \xi = 1.15$

# Summary

- Deep learning based, **grid-free, global solution method** to compute approximate recursive equilibria for discrete-time dynamic stochastic economic models with very high-dimensional state spaces.
- **Key innovation: use the implied error in the optimality conditions as loss function → training data can be generated at virtually zero cost.**
- **comprehensive, scalable and flexible method to address rich models.**

# Questions

